RailClone 2

Parametric Modelling for 3dsMax®

iToo software

Documentation

Image by Bertrand Benoit

Table of Contents

1 1 - Introduction to Next Steps with RailClone 11

1.1 Using this guide 11

1.2 Requirements 12

2 2 - Tips for Creating "Parts" with RailClone 13

2.1 Rule 1 - Model for the generator you intend to use. 13

2.2 Rue 2 - Consider the default mode you are using. Scaled, Tiled, or Adaptive? 23

2.3 3 - Only Model the smallest repeatable components. 26

2.4 4 - Re-use segments using Rotate and Mirror 27

2.5 Rule 5 - Mirror geometry to the other side of the spline. 31

2.6 Rule 6 - Combine segments to create new geometry. Use Compose and Sequence operators to combine geometry and create patterns. 37

2.7 Rule 7 - Building components Individually vs using the Slice modifier? 40

2.8 Rule 8 - Pivot orientation 44

2.9 Rule 9 - Pivot placement 48

2.10 X Alignment 50

2.11 Y Alignment 50

2.12 Z Alignment 52

2.13 Rule 10 - Disguise joints with interlocking geometry 54

2.14 Rule 11 - Reset XForms 55

2.15 12 - Correctly assign material IDs to “parts” 56

2.16 Related Tutorials 58

3 3 - Wiring Parameters and Attributes 59

3.1 Exporting Parameters 59

3.2 Related Tutorials 78

4 4 - How to Control Material IDs 80

4.1 Randomising and Sequencing Mat IDs 80

4.2 UVW XForm Operator 95

4.3 Automatically Box Mapping Segments 108

4.4 Related Tutorials 115

5 5 - How to use Surfaces 118

5.1 Exercise: Using Surfaces with an L1S Generator 118

5.2 Align Y with Surface (L1S Only) 120

5.3 Using Surfaces with an A2S Generator 122

5.4 Extending Boundaries 125

5.5 Related Information 126

6 6 - How to use Clipping Splines 127

6.1 Include and Exclude geometry 127

6.2 Changing Projection Axis 134

6.3 Using clipping splines to define the X/Y Area 138

6.4 Expanding areas - due to padding values or that shapes of certain segment's 140

6.5 Using Extend X/Y Size to Area to create Vertical styles. 142

6.6 Related Tutorials 143

7 7 - Master Spacing Modes 145

7.1 Exercise: Exploring Evenly Spacing. 146

7.2 Exercise: Sync Offset 151

7.3 Exercise: Exploring Default Segment Tiling 153

7.4 Y Mode 157

7.5 Related Tutorials 160

8 8 - How to Fine Tune Corners 162

8.1 Choosing the vertex type. 162

8.2 Turning a Corner Segment's Bend setting On and Off. 163

8.3 Using the Compose Operator with Corners 167

8.4 Bevelling Corners 171

8.5 Automatically Create Curves with Fillet Radius 178

8.6 Tip: Bevelling Corners in Adaptive or Count mode. 179

8.7 Related Tutorials 182

9 9 - Creating Conditional Relationships 184

9.1 Spline based Conditions 184

9.2 Vertex based conditions 195

9.3 Segment based conditions 200

9.4 Using the Selector Operator to create conditional relationships 202

9.5 Related Tutorials 204

10 10 - Creating Presets 207

10.1 Basic Library Usage 207

10.2 Advanced Library Usage 215

10.3 Running Maxscript functions 221

10.4 Installing Libraries 223

11 11 - Next steps 226

11.1 Tutorials 226

11.2 **Forum** **226**

11.3 **Documentation** **226**

11.4 **Gallery** **226**

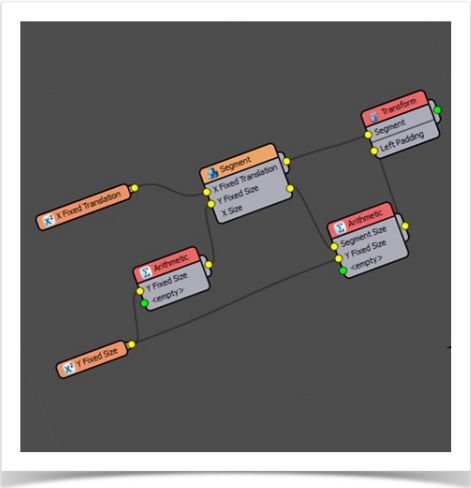
If you've already read Getting Started with RailClone and you're keen to learn more, then you've come to the right place! In Next Steps with RailClone we build on the foundational concepts learned in the first guide and introduce techniques to take your parametric modelling to the next level. It is our intention that by the time you reach the end of this document, you will have covered nearly all the features that RailClone offers and you'll be ready to make it an indispensable part of your your day-to-day workflow.

This guide features 11 themed chapters.

A PDF and EPUB version of this guide for offline viewing is also available.

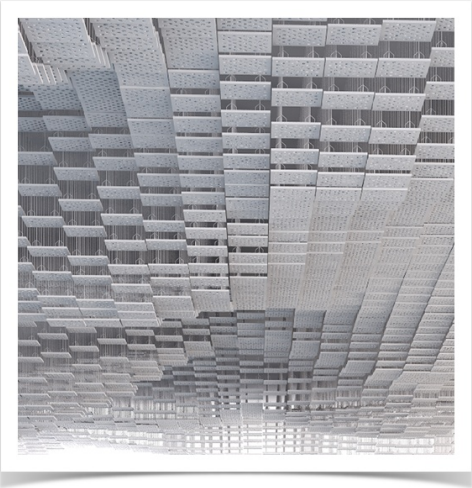
Chapter 1 - Introduction

Chapter 2 - Tips for creating "parts" for RailClone



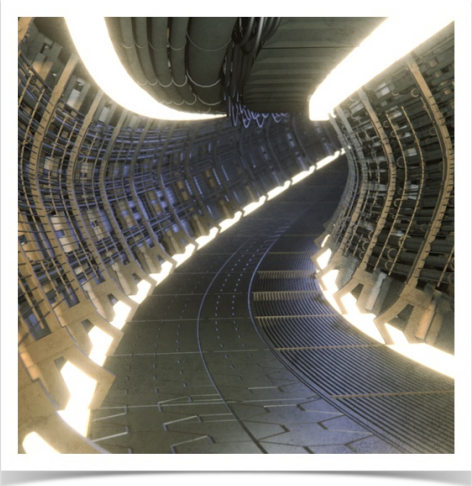
Chapter 3 - Wiring Parameters and Attributes

Chapter 4 - How to control Material IDs

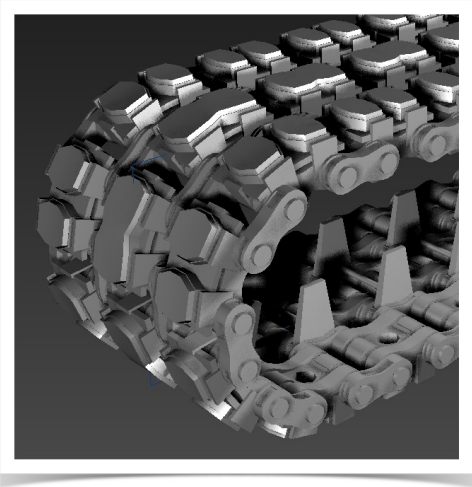


Chapter 5 - How to use Surfaces

Chapter 6 - How to use Clipping Splines

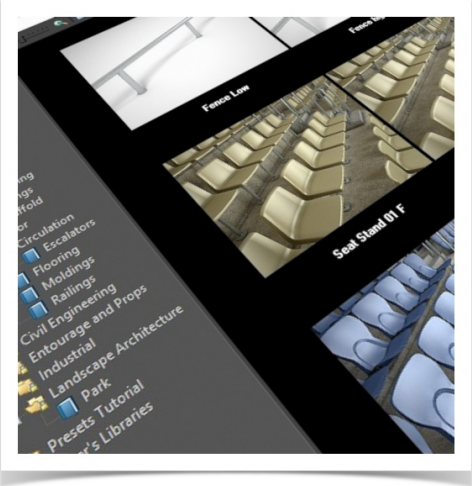


Chapter 7- Master Spacing Modes

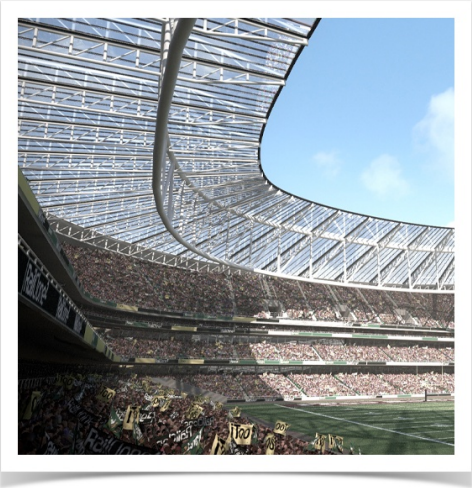


Chapter 8 - How to fine-tune Corners

Chapter 9 -Creating Conditional Relationships



Chapter 10 - Creating Presets



Chapter 11 -Next Steps

1 1 - Introduction to Next Steps with RailClone

If you've already completed Getting Started with RailClone or you've been using RailClone for a while and you want to dig deeper into more advanced features, then this guide is for you.

Over the next 10 chapters we'll look at how to approach modelling for RailClone, with a focus on the key elements to RailClone's internal logic that affect the geometry creation process. We'll also look at how wiring works to help you build relationships between parts of a RailClone graph and make styles much easier to use and re-use. We'll show you how to create libraries so that you can share your creations with a wider audience, either in the same studio or online, and we'll tame tiling modes and corners, introduce you to expressions, controlling material ids, and creating and editing UVW mapping. To round off we'll look at how you can use clipping splines to create arrays of any shape, deform RailClone objects to follow surfaces and create conditional relationships and much more.

If you have any questions or comments about the contents of this guide, please feel free to post them on our forum.

1.1 Using this guide

We recommend working through this guide from start to finish. It has been structured to that each new section builds on the concepts learned in the last chapter. To help you get more from the guide, each new chapter contains the following sections:

Exercises

We've included even more exercise in this guide to help you develop the skills you need to use RailClone for many of the relevant day-to-day tasks you might encounter. We recommend taking the time to go through these exercises, but we're aware that time is precious so the exercises have deliberately been kept short and simple. In most cases there is a start and end version of the file, so you can jump ahead and see how the final result should look.

You can download the source files for the exercise below. The .max files are in 3ds Max 2010 format with standard materials.

Click here to download the exercise files for Next Steps With RailClone.

Most the exercises can be completed with RailClone Lite, but occasionally Pro is required. Where an exercise requires RailClone Pro you'll see this symbol PRO only .

Relevant Tutorials

At the end of each chapter you will find links to recommended tutorials. Follow these to learn more about each subject, see the principles put to use and learn more advanced skills.

Links to documentation

To find out more about a subject, each chapter provides links to the relevant areas of RailClone's online documentation. If you have a question, there's a good chance you'll find the answer there.

1.2 Requirements

Though it isn't essential if you're new to RailClone you should certainly read the Beginners Guide. It explains many core concepts that are not repeated here. Apart from that all you need is a working knowledge of 3DS Max and the most recent version of RailClone Lite or Pro!

If you're ready, let's get started by looking at how to create source geometry to be used with RailClone in Chapter 2 - Tips for Creating "Parts"

2 2 - Tips for Creating "Parts" with RailClone

RailClone's approach to creating parametric objects is unique. By instancing, sequencing and combining modular, repeatable components of geometry; objects featuring huge polycounts that render with no performance overheads and rapid iteration are easy to create.

If you've been using RailClone for a while, you'll doubtless have experimented with the built in library styles; however, one of the most important steps in moving from using RailClone's built in library styles to creating and using your own, is understanding how to create modular geometry that works well with RailClone's two array types. This subject has already been touched on briefly In the Getting Started with RailClone Guide where we outlined a few rules for creating geometry for L1S and A2S generators. In this chapter we'll recap and expand upon that foundation and look at 11 useful guidelines that can help make creating geometry for RailClone even easier.

2.1 Rule 1 - Model for the generator you intend to use.

The most important thing to consider when creating source geometry "parts" to be used by RailClone is the type of generator you intend to use. To do this you'll need to to examine the design you intend to create, and analyse how it will fit into one of the two array types. To do this it's obviously important to have a good understanding of RailClone's rule-based work flow, so if you need a recap we recommend you check out Key Concepts: Array Based Instancing from the Getting Started with RailClone Guide.

Once you've ascertained the most suitable array, you'll have a list of the possible inputs that can be used. For example, if you're planning to use an L1S generator you have 5 possible inputs, while the A2S generator has over twice as many with 12 possible inputs. These are combined in the arrays in rows and columns as shown in the table below. Remember the **default** rows and columns are repeated or scaled in between the other segments to fill in the array.

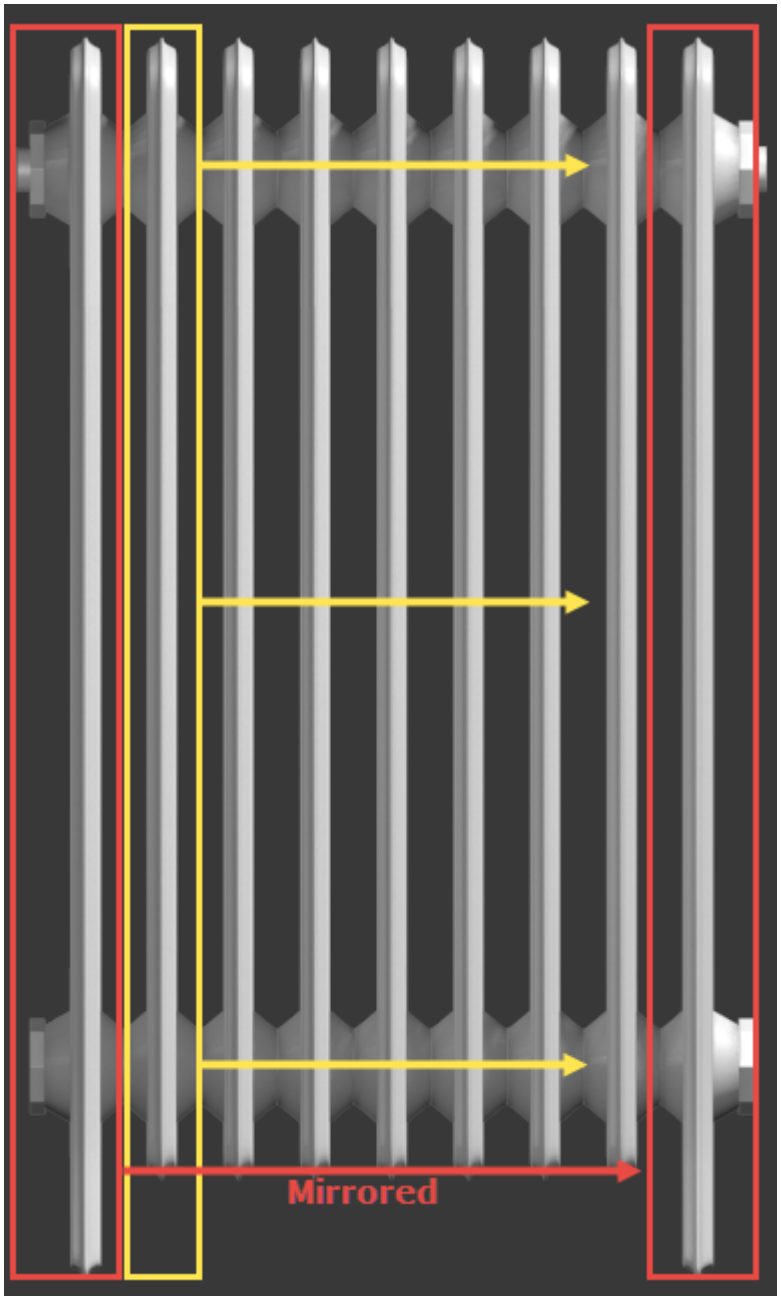
L1S Array	**Start**	**Default**	**Evenly**	**Corner**	**End**
A2S Top Row	Top Left Corner	Top	X-Evenly/Top	Corner/Top	Top Right Corner

A2S Y Evenly Row	Left /Y-Evenly	Y-Evenly	X - Evenly	Corner	Right/Y-Evenly
A2S Default Row	Left	Default			Right
A2S Bottom Row	Bottom Left Corner	Bottom	X Evenly /Bottom	Corner /Bottom	Bottom Right Corner

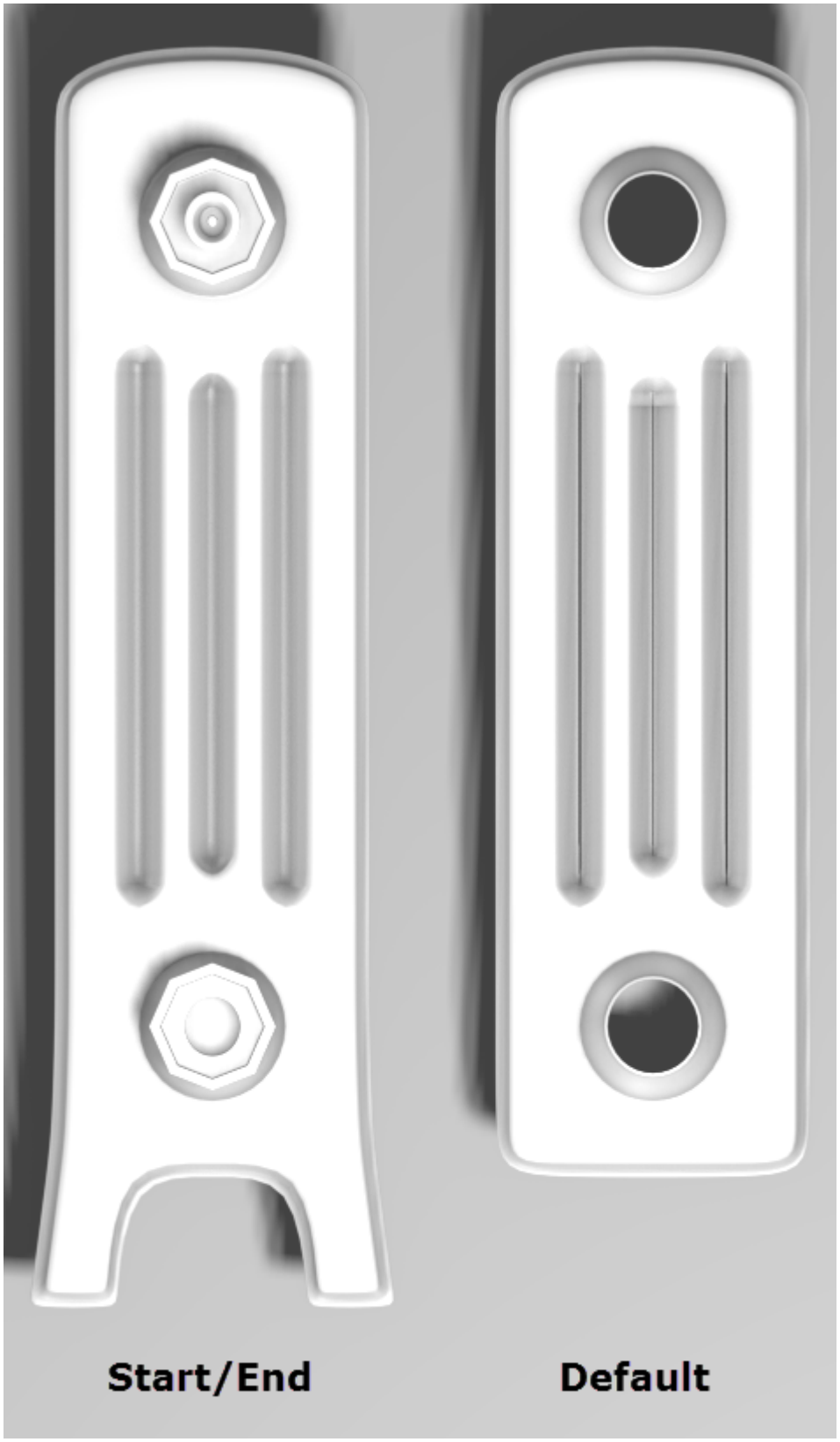
Though there is a wealth of inputs to help you create your designs, there are also a few limitations that you should be aware of. For example, in the A2S array, there isn't an input to target the ends of a Y evenly Row or an X Evenly column; there's also no input for the intersection between the Y evenly row. and X Evenly columns either. Of course it's possible to get around this limitation using conditional statements and other tricks, but it's still an important consideration to bear in mind when you start to create your assets.

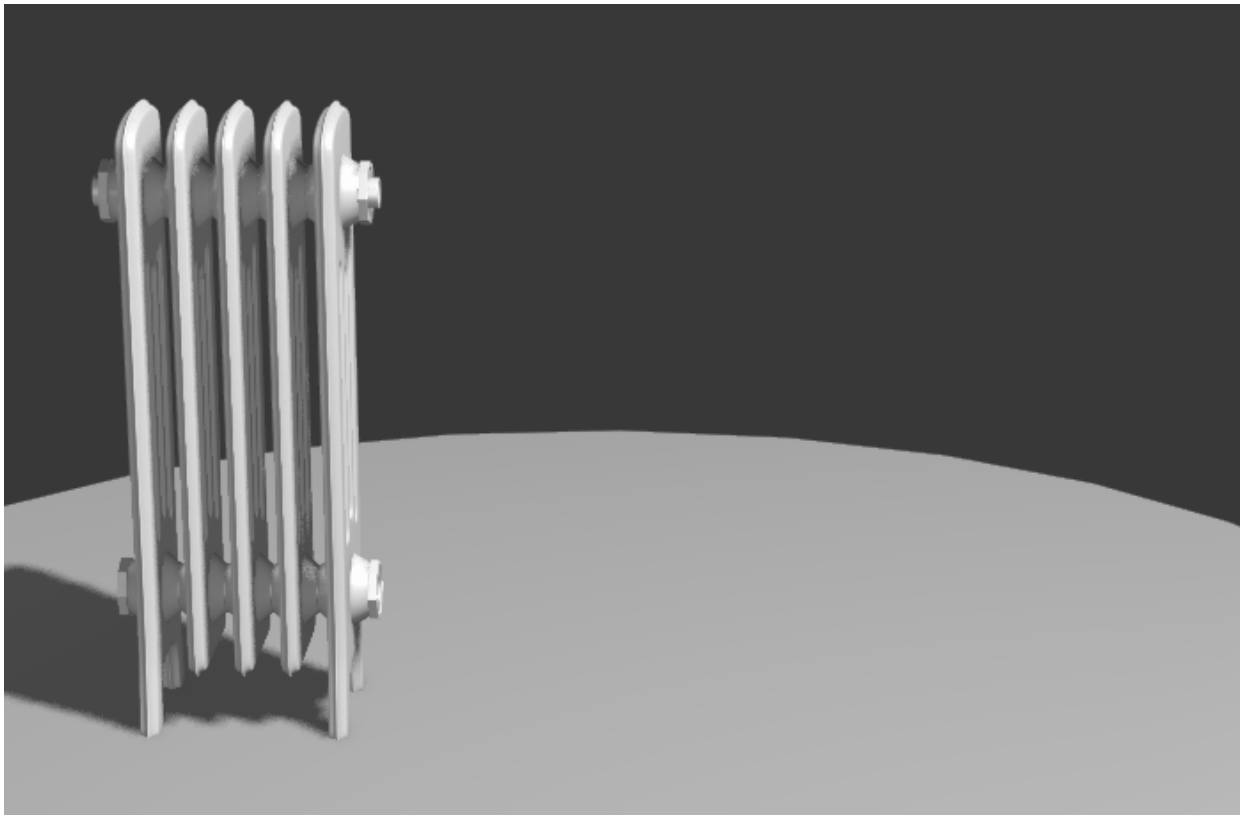
In many cases you will be have the choice of either of the two generators to create a style, when this is the case, the decision of which array to use can be determined by questioning how many dimensions you need to adjust in the style. For example, let's imagine you have a job that specifies a particular style of radiator as seen in the screen grab shown below.

If all the radiators in the project have the same height but different lengths you can use the simpler L1S - linear generator. This is generally much easier to create source geometry for. In this case you can re-create the style with only two segments. The image below shows how you would analyse this design.



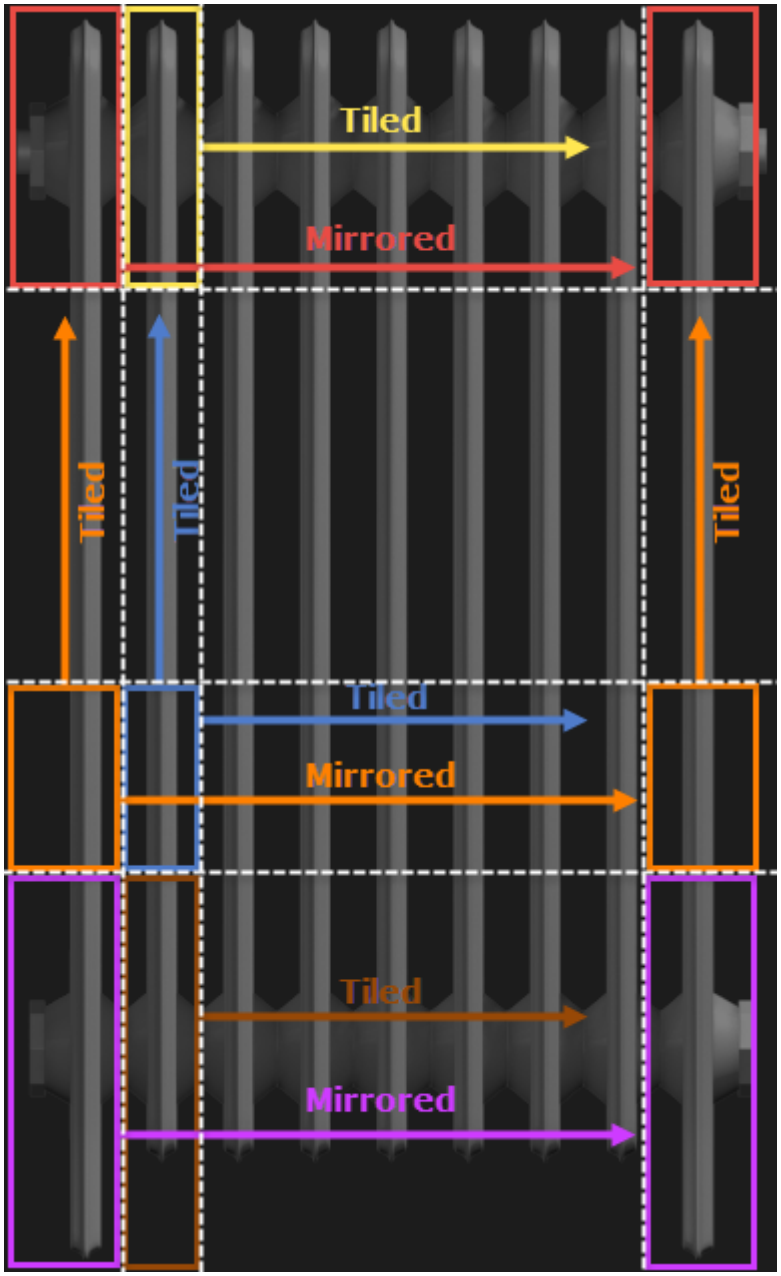
As you can see there's a repeating radiator fin section that is marked in yellow. In an L1S array the segment wired to the Default input is repeated to fill in between Start, End, Corner and Evenly segments, so we can model the area marked as yellow to create our Default segment's geometry. This will be adaptively tiled between a Start and End segment that incorporates the radiator's legs and valves, marked in red. There's no need to model this twice to create separate models for the Start and End, as we can simply use a Mirror operator to flip and reuse a single segment. Based on this analysis we only need the following two segments to create a radiator of any length.


Start/End
Default



However, if the radiators are of different lengths **and** heights you can use an A2S generator to create an adjustable two dimensional array. To use this generator the geometry will need to be prepared quite differently and when we analyse the design, this time we need to consider how to break up the radiator to fit into rows and columns.

Firstly let's look at the columns, the same principle applies as we used to analyse the design for the L1S array. There's a Start column, mirrored to create the end, and then repeating fins in-between. But that's not all, we also need to consider the Y axis. The top and bottom have distinctive geometry which cannot be mirrored so we'll also need separate segments in each column for the top and bottom of the array. We'll also need a default row which tiles in between the top and bottom rows as the height is adjusted. The image below illustrates how to approach analysing this radiator design to create models for the A2S array.



This pattern is derived from thinking about the model as a series of horizontal and vertical slices that create columns and rows. By slicing the model as shown by the white dashed lines we get the following segments, organised by row:

Top Row

The **Top Left Corner** and **Top Right Corner** segments are shown in red. Only one segment is needed because we can use a mirror operator to create the other side
The **Top** segment is shown in yellow, this will be tiled on the X axis to fill in between the Top Left Corner and the Top Right Corner.

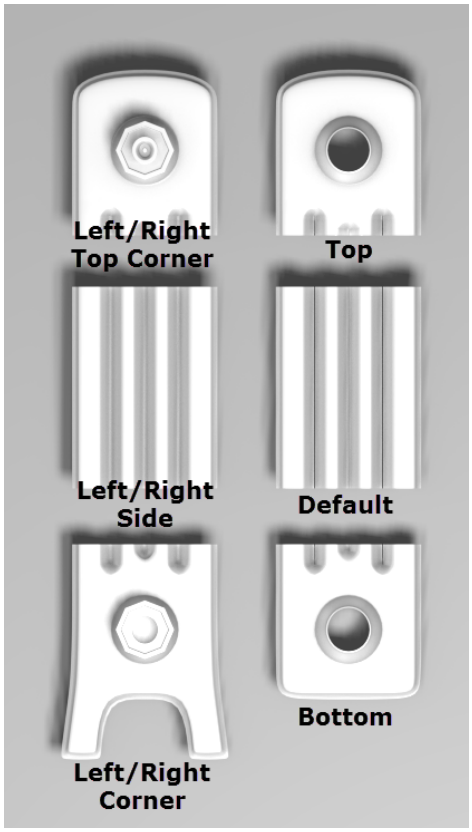
Default Row

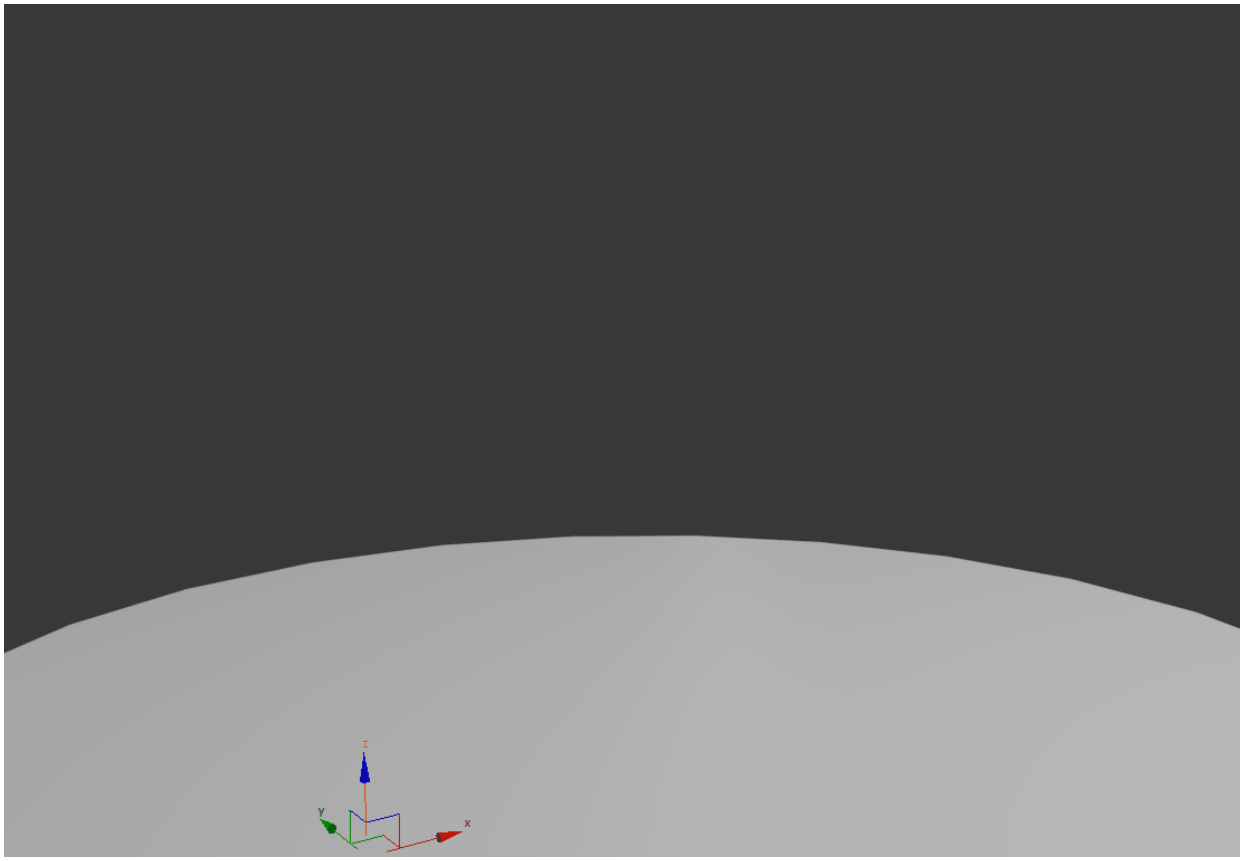
The **Left** and **Right** segment are shown in Orange, Only one segment is required as we can mirror it to create the other side. This segment will tile on the Y axis to fill in between the Top and Bottom rows
The **Default** segment is shown in blue. This tiles on the X axis to fill in between the Left and Right segments, and on the Y axis to fill in between the Top and Bottom rows.

Bottom Row

The **Bottom Left Corner** and **Bottom Right Corner** segments are shown in magenta. Like the top corners, only one segment is needed because we can use a mirror operator to create the other side
The **Bottom** segment is shown in brown, this will be tiled on the X axis to fill in between the Bottom Left Corner and the Bottom Right Corner.

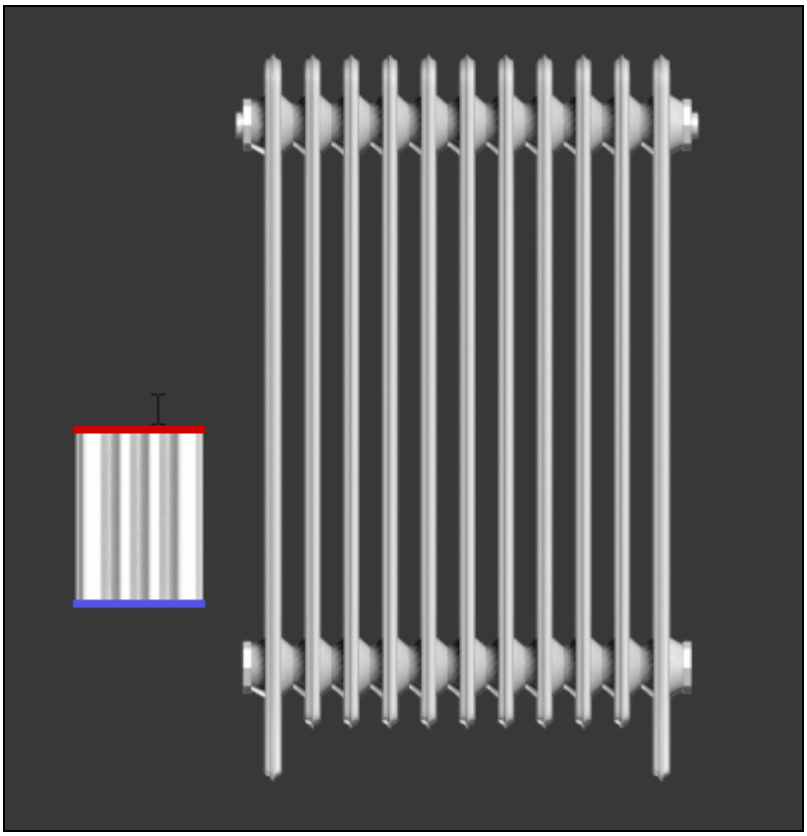
Based on this analysis we can model the following six segments to create a radiator of any length and height. To see the finished file, open *chapter_2_radiator.max* from the downloads for this guide.




Z
Y
X

Segment Height in A2S Arrays

If you are creating a 2D array with no gaps, then It's important to ensure that the height of all the segments in a single row is the same. The height of a row is determined by the bounding box of the largest segment and rows cannot automatically overlap. In the example below you can see what happens to the radiator style if we adjust the height of the Side segment so that it does not match the Default segment.



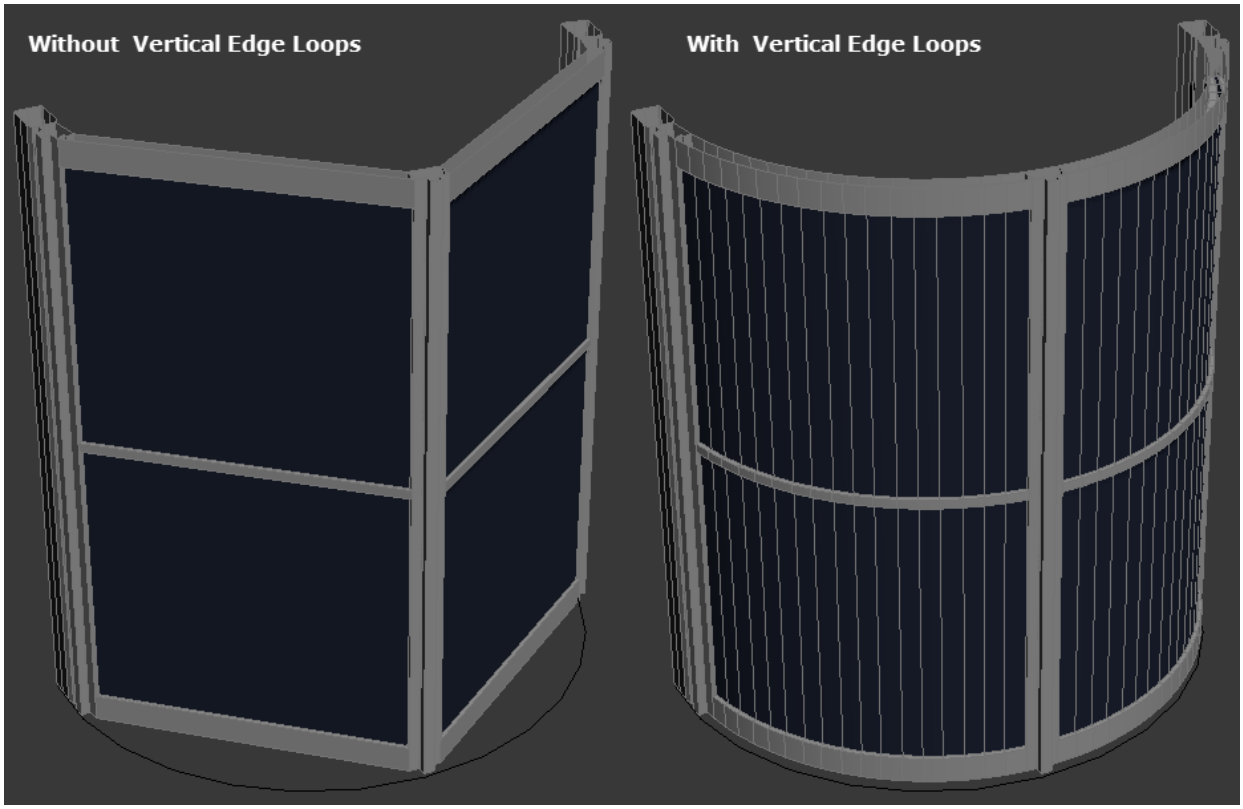
As you can see from these examples, the key to creating usable assets for RailClone is dependant on a good understanding of the inputs of the two types of generator. If you'd like to look at some examples, please feel free to reverse engineer our built in library objects. The geometry for each of the segments can be extracted into the scene to be studied or edited by selecting the node and clicking from the toolbar in the **Style Editor**.

2.2 Rue 2 - Consider the default mode you are using. Scaled, Tiled, or Adaptive?

When you are creating geometry to be used in the Default. Bottom, Top or Y Evenly inputs, it's helpful to keep in mind the **Default Segment Mode** you intend to use. This set of options, which can be found in the generator's **Properties** > **Rules** panel, determine the way in which segments are distributed along a path section. In the default Tile mode, the segment is simple replicated along the path. If the last segment in the section is too long, it is sliced to fit the path's length. This is the most forgiving of the Default modes, and more or less any geometry will be fine as long as you don't mind it being sliced at the end of the path.

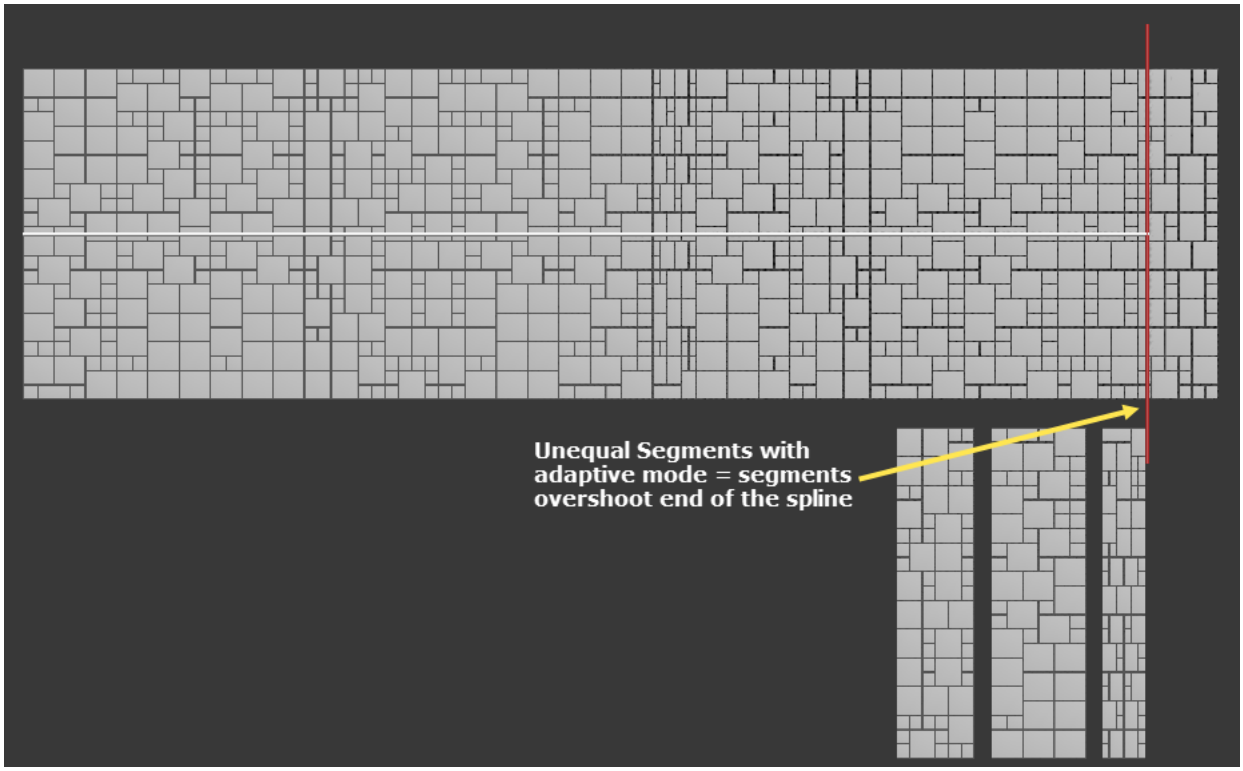
Instead of tiling you can also use **Scale** mode to stretch a single segment to fit the path section. Because they are stretched, segments for use with **Scale** mode will generally require a consistent cross section to avoid unwanted distortion. This mode is well suited for situations where you might normally be tempted to use the Sweep modifier. To create an cross section for this mode, the easiest way is often to draw the cross sections in the Left or Right viewport with spline tools, and then add an extrude modifier to create polygonal geometry.

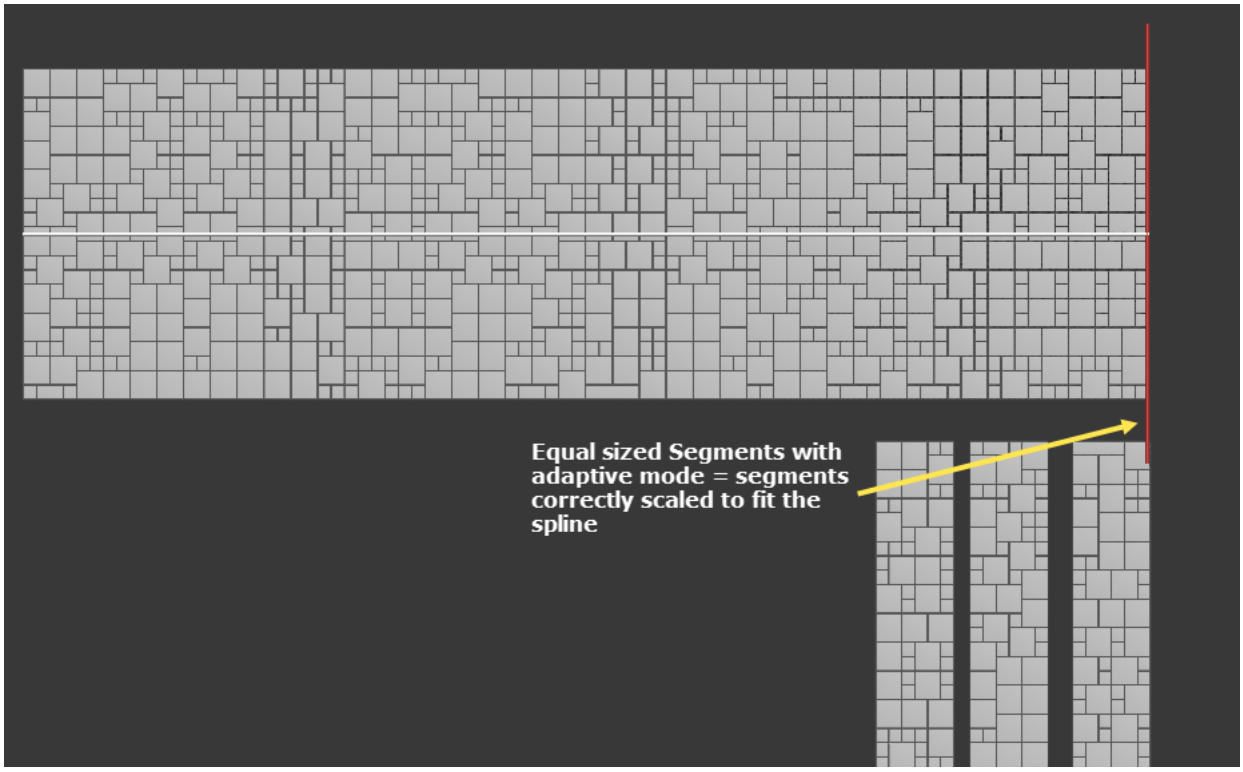
Another important consideration when creating geometry for Scale mode is whether of not you would like the geometry to follow the curves of the path or travel in a straight line from the start of the path section to the end. If you don't include any vertical edge loops, the geometry will scale in a straight line, however including vertical cuts will allow the segment to deform to follow the path.



Count mode allows you to specify an exact number of whole segments to use between the path start and end. Depending on the number specified this can also cause stretching along the X axis so be sure to only use geometry where that is expected and acceptable.

Finally, **Adaptive** mode is similar to Tile except instead of slicing the final segment, this mode calculates the number of whole segments that would fit along a spline segment then subtly scales them on the X axis to ensure they fill the available space. Any distortion on the X axis is generally quite small, so you can get away with modelling as you would when using Tile. There are very few restrictions when modelling for Adaptive mode unless you are planning to use multiple default segments on a single path section, for example by using the Compose, Sequence or Randomise operators. If you do use default segments with different lengths, you could find that the final segment sometimes overshoots the ends of the path. The Adaptive algorithm is only able to accurately scale the meshes if they are identically sized on the X axis, so if you plan on using this mode with multiple segments, ensure they are all the same size.



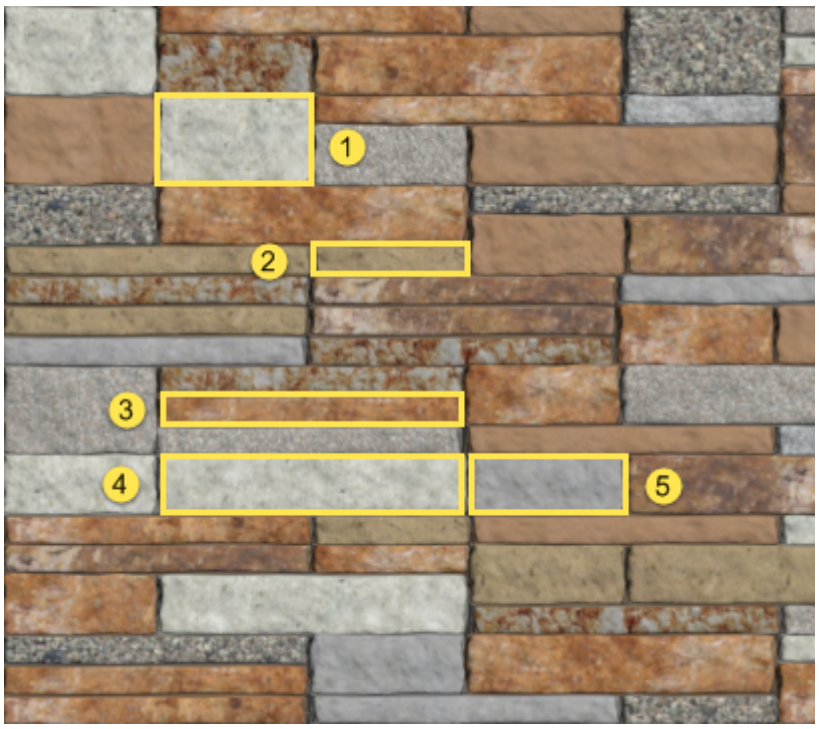


We will look at tiling modes in more detail in Chapter 7 - Master Spacing Modes of this guide.

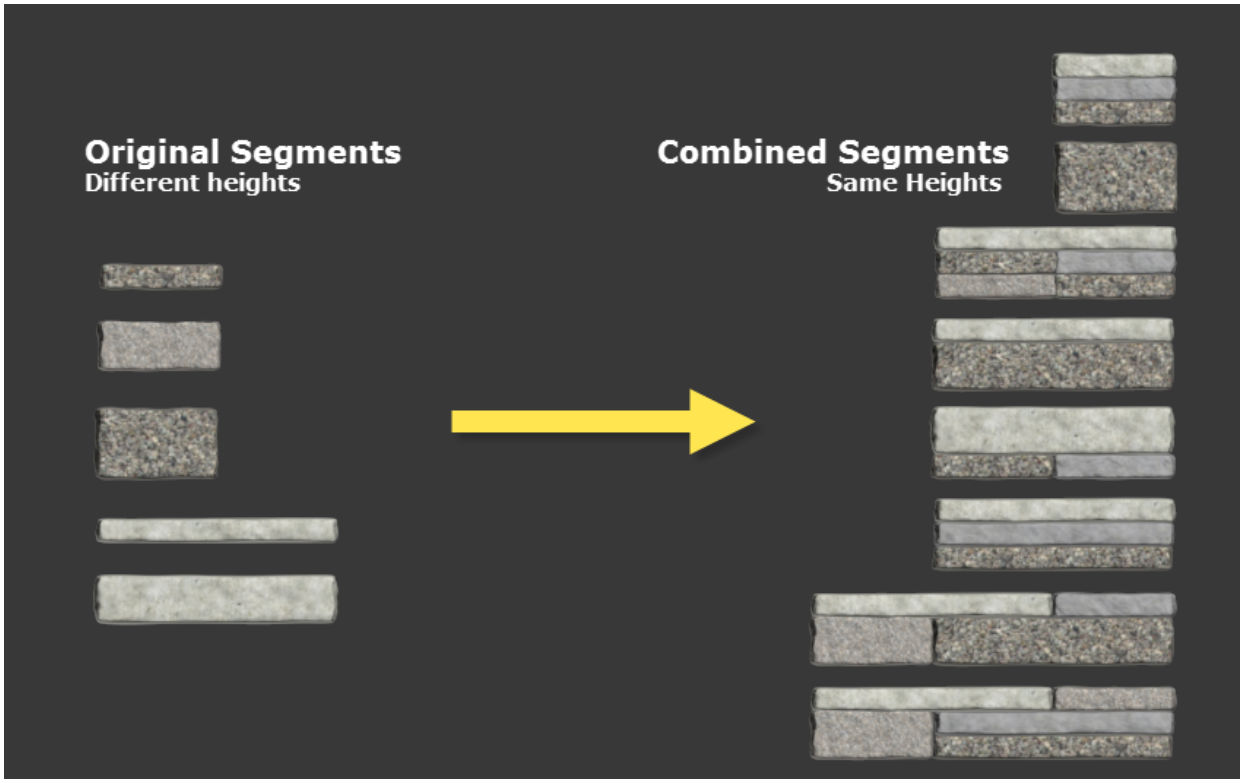
2.3 3 - Only Model the smallest repeatable components.

To use RailClone efficiently, you should try to identify the smallest repeatable modules of geometry, and create only them. There's no point in spending valuable time modelling geometry that is easier to generate automatically using RailClone. By carefully examining your designs, as in the radiator example above, you can identify the minimum number of components required to create the style. However as we've also seen, the critical consideration is that the parts should fit into one of RailClone's 2 array types. In some situations, you may be able to identify repetition in a design that does not easily fit into a L1S or A2S generator. For example, take a look at this ledge wall example:

We can identify 5 small repeated components, shown highlighted below.



However because there are 3 different height of the stones they won't fit into an A2S array without creating gaps, (Remember the A2S array requires all the segments in the array to be an equal height). Although these are the smallest elements, we'll need to think a little more about the style to make it work with RailClone. To fix this, Instead of using individual stones, we can create combinations of stones until they are all the same height, and then add these to RailClone. With these 5 stones there are a number of possible variations as show below.



These aren't the smallest components in the wall design, but they are the smallest that will fit easily into the A2S array.

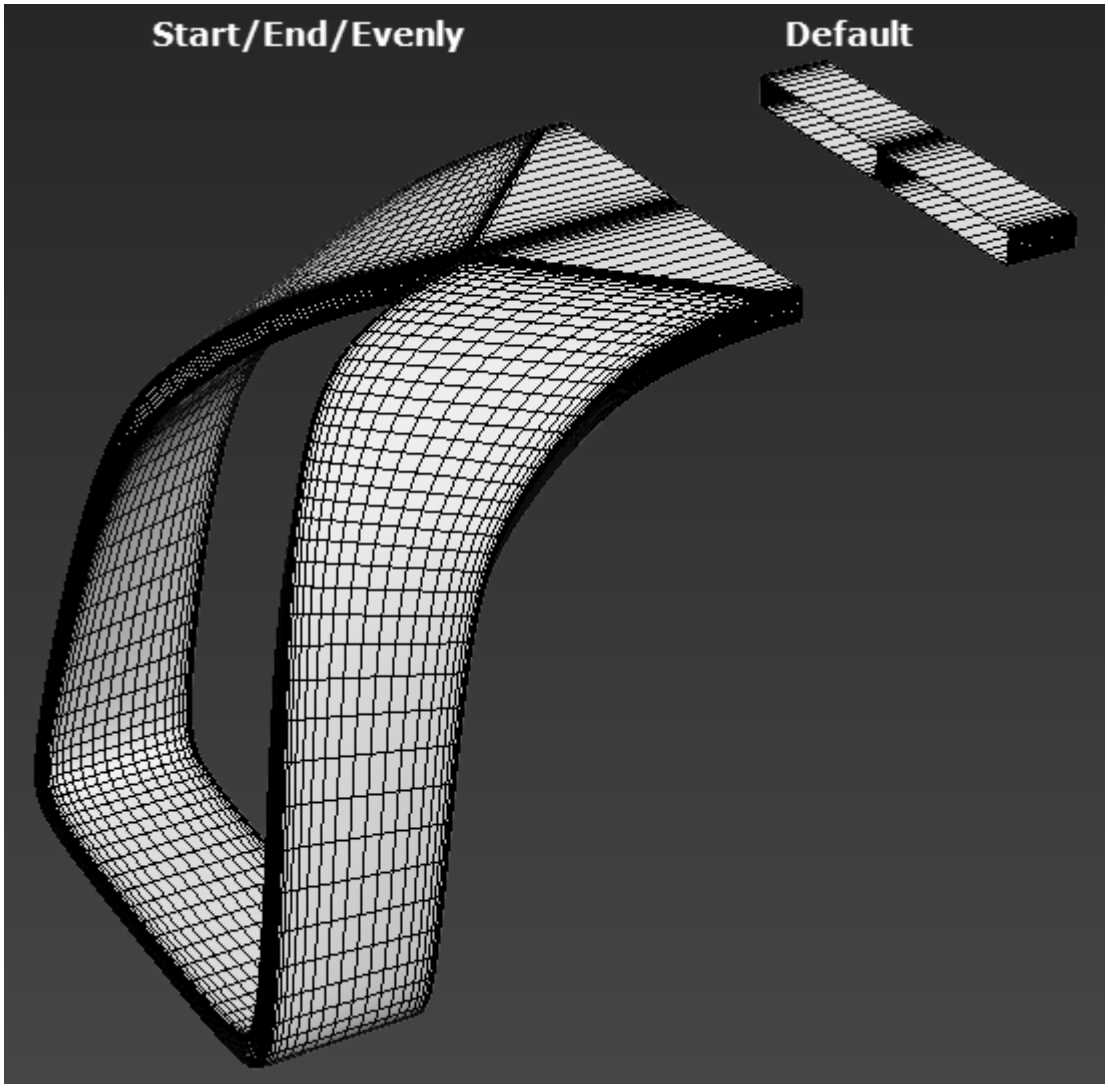
So, despite the heading title, smaller isn't always better. There's a balance to be found between the polygon count of the source geometry and the number of segments that are created in normal use. Another consideration is if a segment has a very high poly-count and is likely to be sliced or deformed. In this case it will no longer be instanced and contributes to the total amount of static geometry in the scene. Therefore if you know that many of the segments are going to be sliced or deformed you should try to keep the polygon count of each individual segment relatively low. This trade-off between the number of instances and the polygon count of segments that will be sliced or deformed is the key to producing well optimised styles. There's no fixed figures for this, it depends very much on your scene and chosen renderer. We recommend you do some tests if the style is likely to produce many millions of instances.

2.4 4 - Re-use segments using Rotate and Mirror

This is very closely related to the previous points. As well as looking for elements in the design that are repeated, try to look for geometry that can be re-used by rotating or mirroring on the X, Y, or Z axis. RailClone includes the ability to transform or flip geometry making it even easier to reuse.

For example, take a look at this bench design by BMW Group Designworks USA.

This can be created in RailClone using only 2 segments. The metal ends of the seats are unique and cannot be broken down any further, whereas the wooden seat itself is a simple profile that can be scaled along the length of the bench. Because the end can be mirrored there's no need to create a segment for each end - just use the mirror operator. The two segments to recreate this style would look like this:

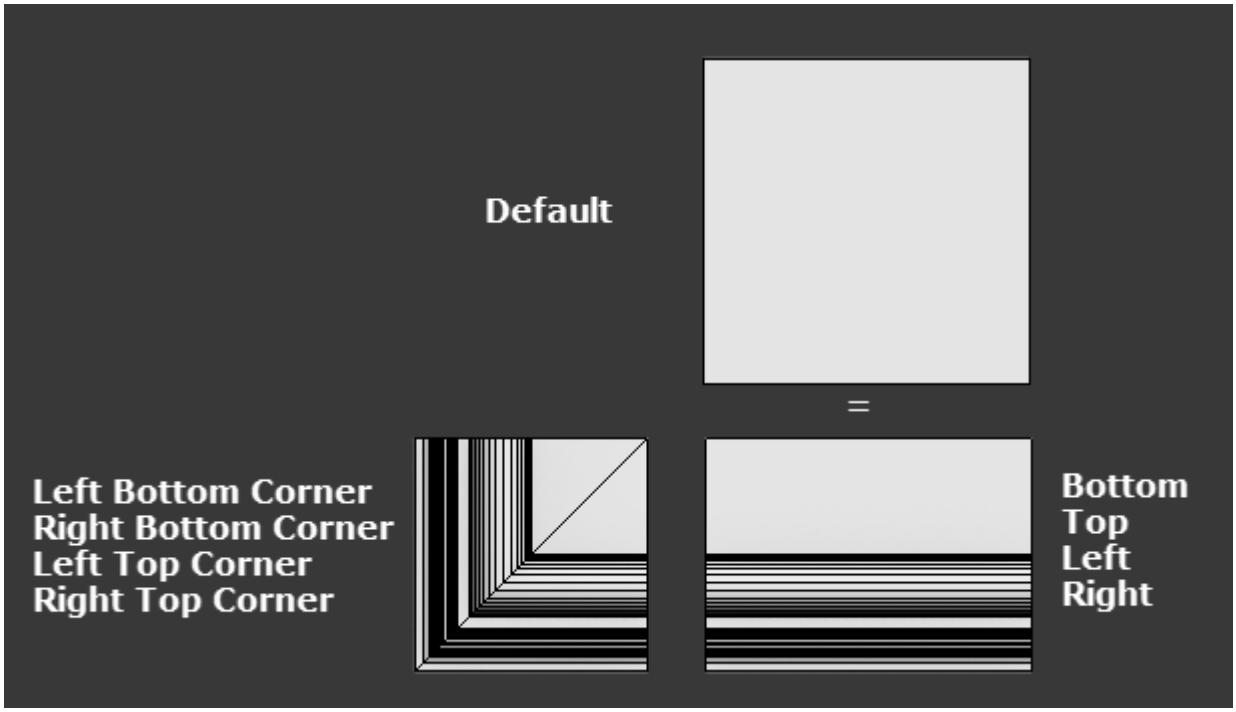


See the chapter_2_bench.max file for an example of this finished scene

In the radiator example above we also used the same technique, by modelling the segments for the left hand side of the radiator and mirroring them, we were able to create the segments for the right.

When you're using the A2S array you have even more opportunities to re-use geometry. For example, let's look at the geometry required to make a simple picture frame with an adjustable height and width. If we slice up the the model we can see that there are 9 segments required to make this style work.

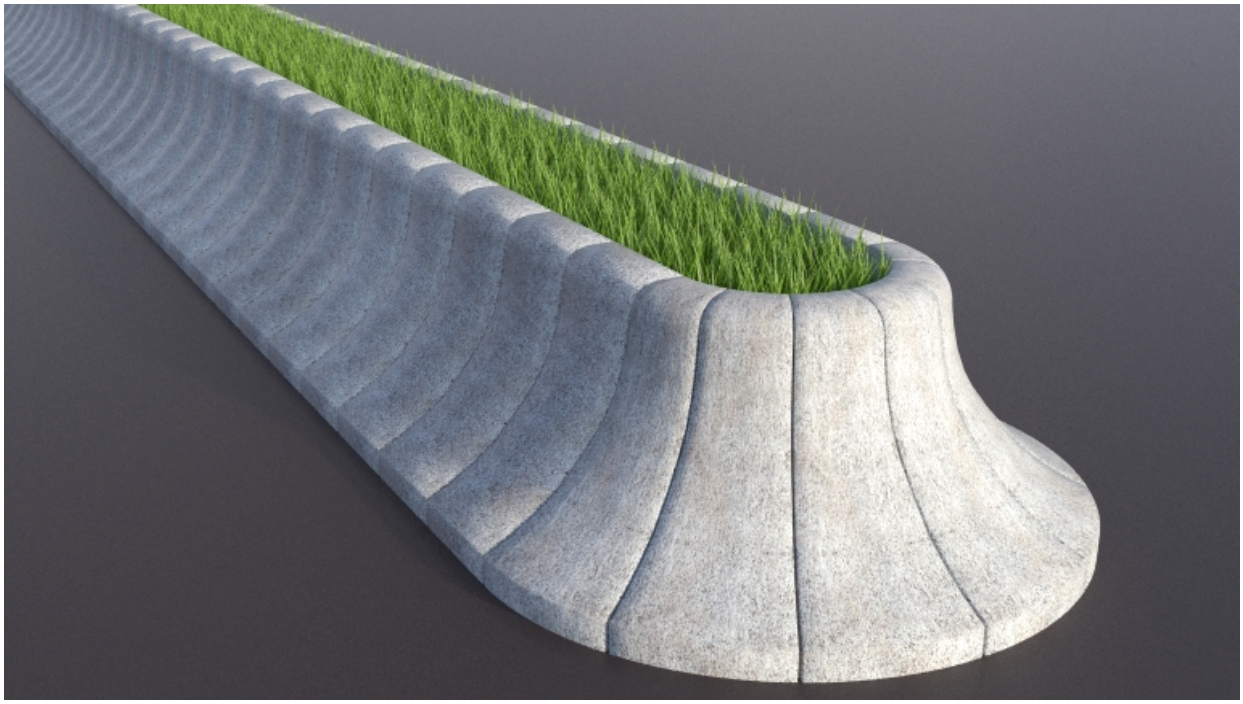
9 pieces is a lot for such a simple style, fortunately it can be created with far fewer. In the diagram above there are 4 separate corners segments, but by mirroring the bottom left corner segment on the X axis we get the bottom right, mirror on the Y axis to get the top left, and finally mirror the segment on the X and Y axis to get the top right. In this way the same segment is used 4 times to fill the corners of the the array. The same is true of the bottom of the frame which can be mirrored on the Y axis to create the top and rotated by 90 degrees to create the sides. By reusing segments with mirror and rotate we're able to reduce the number of source objects from 9, to 3.



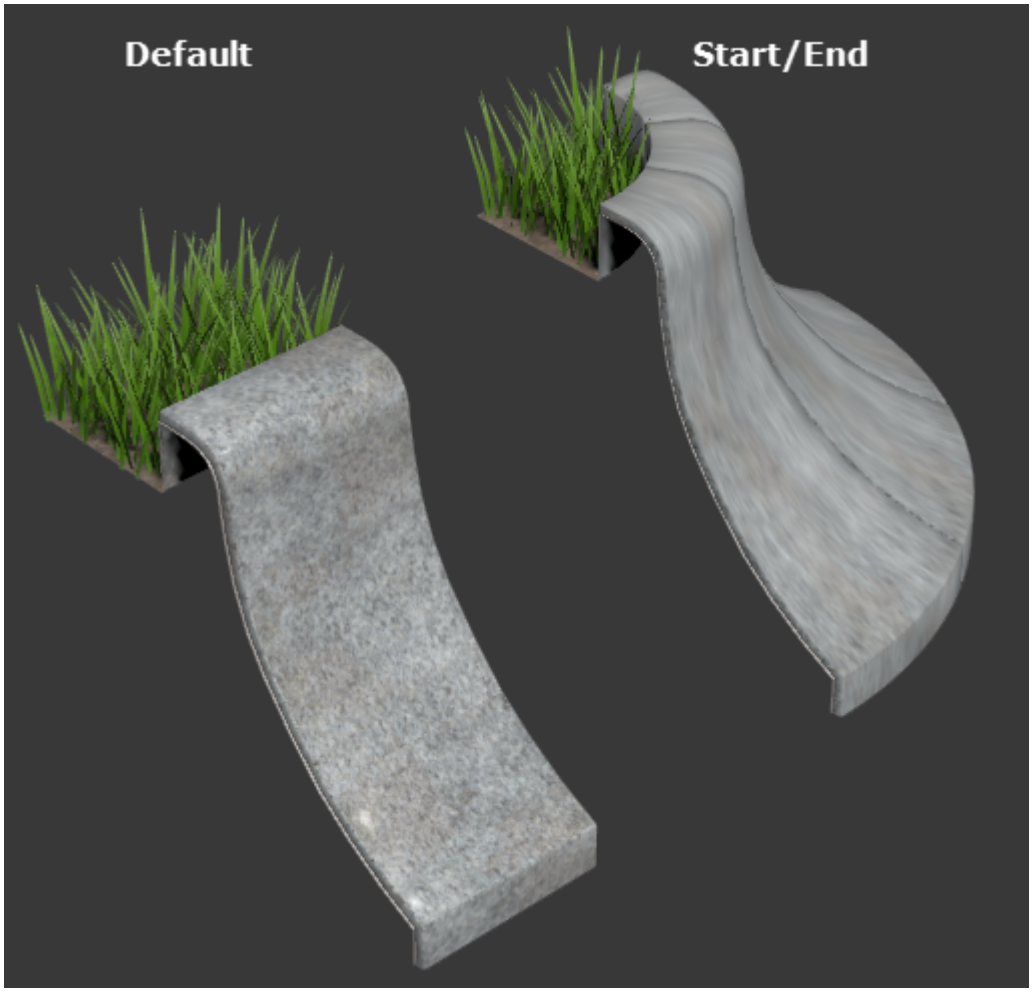
To see this style, download *chapter_2_picture_frame.max*

2.5 Rule 5 - Mirror geometry to the other side of the spline.

In the previous example we looked at how you can save modelling time by mirroring or rotating segments so that they can be re-used. In addition to the ability to use the Mirror operator to flip individual elements, the L1S generator also features the ability to duplicate and mirror the output of an entire generator either side of the spline. Take for example the *Median* style from the built in library:

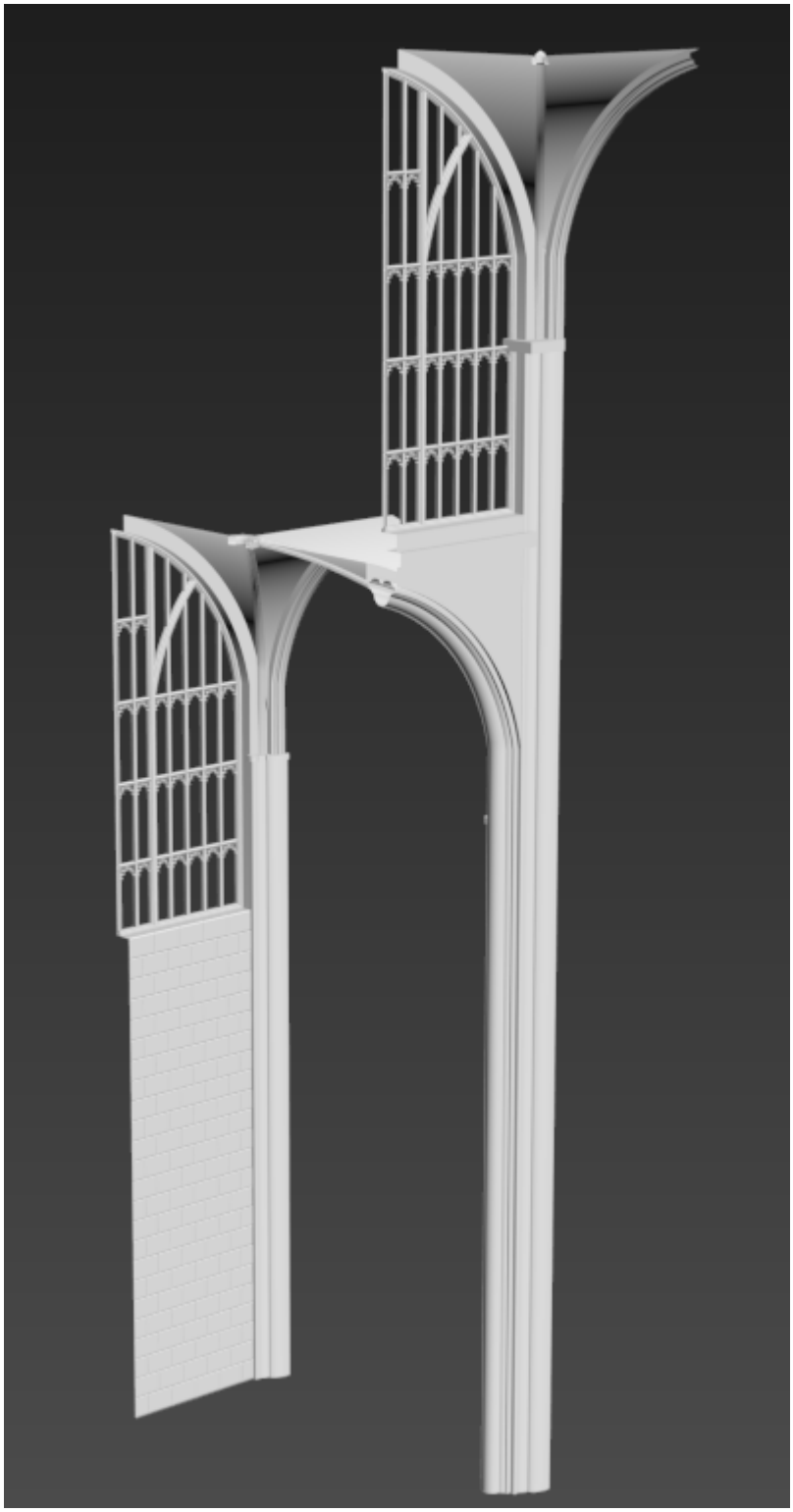


This simple style uses 2 segments, a Default and an End, but as you can see, we've only modelled one half of the median:



By exploiting symmetry in this way you are able to reduce the amount of modelling necessary to create the style. Let's look at another example that uses an L1S Generator with Symmetry and a Mirror operator to make the amount of modelling necessary to create the style minimal.

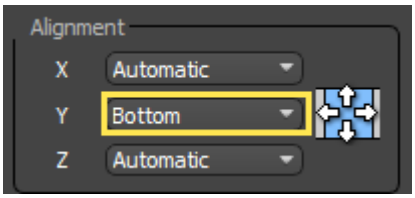
In this second example, imagine we're blocking out a Gothic style interior. We can create a segment like the image below - it looks like a fragment but by using the mirror operator, a sequence operator and turning on Symmetry for the generator we can create an entire interior.



Exercise: Using Mirroring Tools

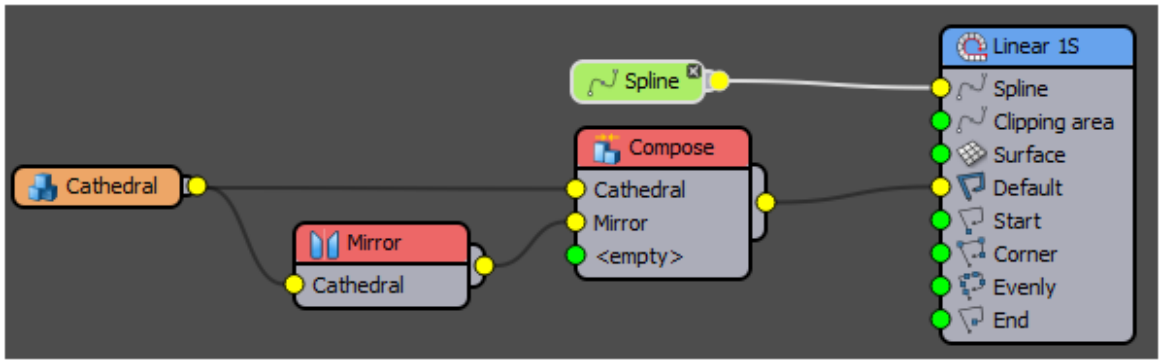
1. Open *chapter_2_symmetry.max*.
2. Select the *rc_cathedral* RailClone object.
3. Open the **Style Editor**, the Cathedral segment has already been added to the style.

4. Select the segment, go to the **Properties** panel and change **Y Alignment** to **Bottom**

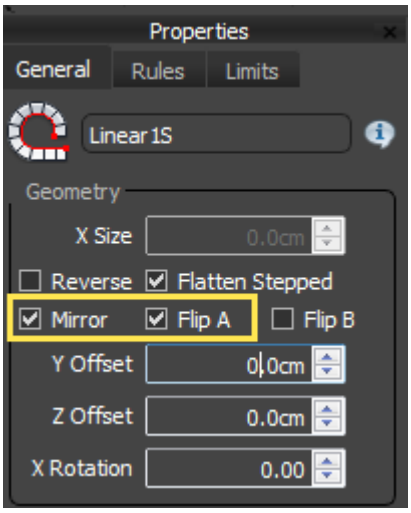


This will move the segment so that its aligned to one side of the the spline instead of its centralised.

5. Create a new **Compose** operator. We'll use this to alternate between the normal and a mirrored version of the segment along the X axis.
6. Wire the Segment to the **Compose** operator's **first** input.
7. Create a new **Mirror** operator and wire this to the **Compose** operator's **second** input.
8. Wire the Segment to the **Mirror** operator's input. You're node tree should now look like this:



9. You'll now have half a Cathedral, to create the other half select the **Generator**, open the **Properties** and turn on **Mirror** in the **Geometry** group. The geometry is copied to the other side of the spline, but it hasn't been flipped. To do this turn on **Flip A** to complete the style.

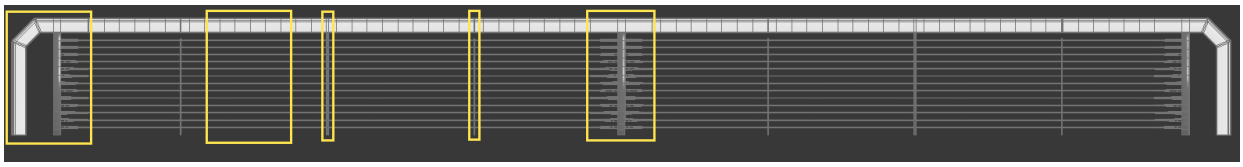


In this example we maximised the reuse of the segment by mirroring it on two axes. The great thing about working with so little geometry is that making design changes is very fast. At the moment this Cathedral is pretty simple, but if we wanted to start modelling additional detail, re-texturing, or adjusting the proportions, we only need to change the one segment and all the changes will cascade through the entire style.

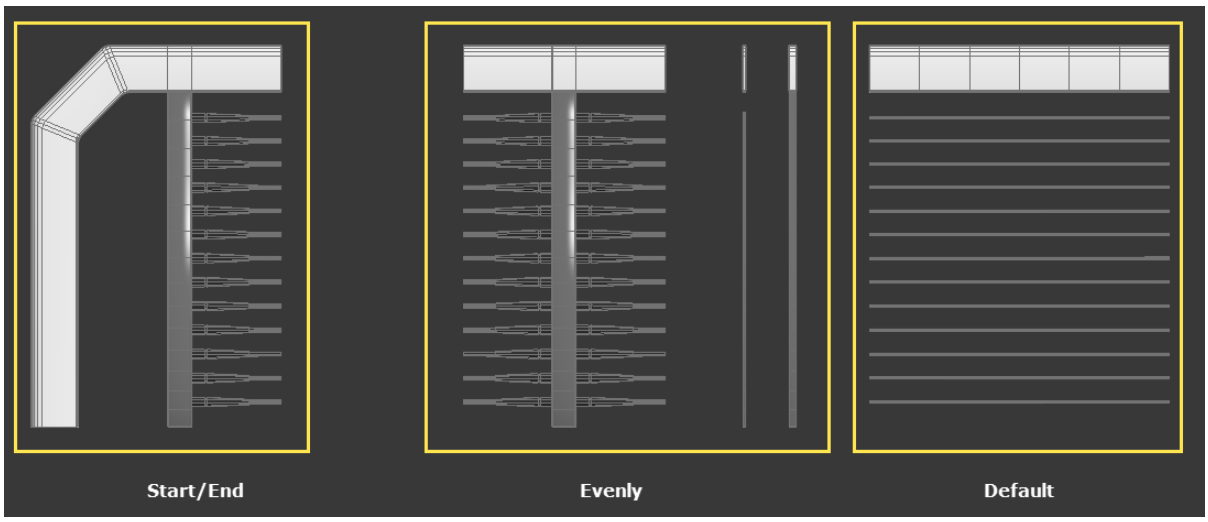
2.6 Rule 6 - Combine segments to create new geometry. Use Compose and Sequence operators to combine geometry and create patterns.

Save time modelling geometry by finding opportunities to combine segments. RailClone includes a Compose operator that enables you to add two or more segments together to create a new component. Combined with the Sequence and Reverse operators you can create new geometry by combining smaller units of geometry found elsewhere in the design.

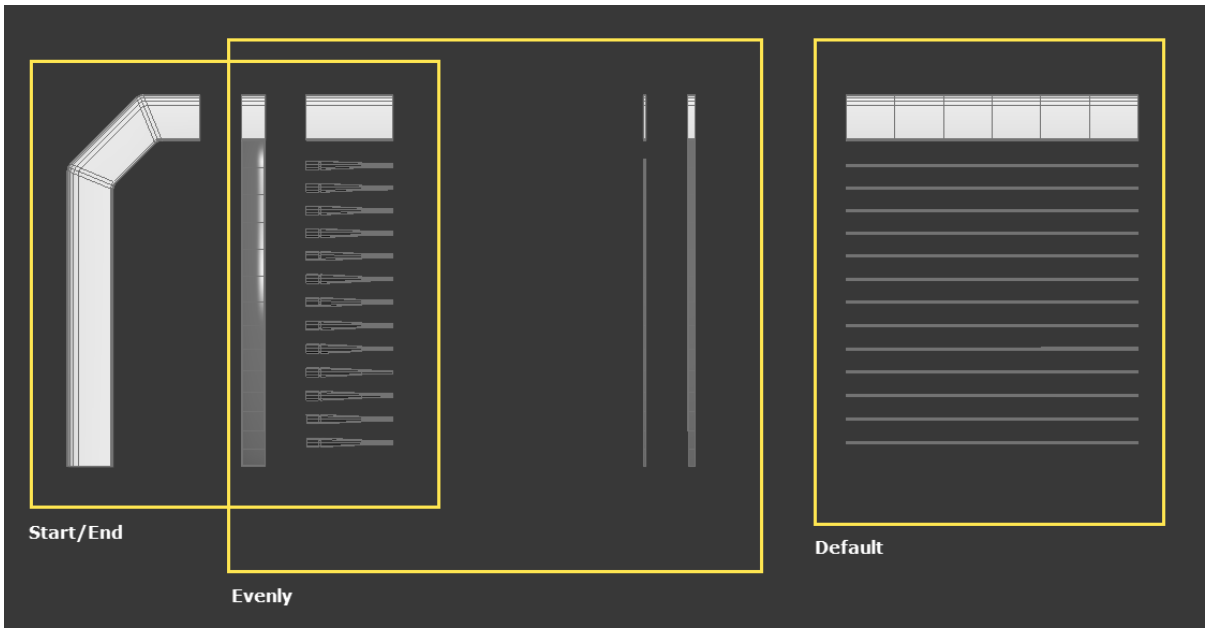
Consider this railing for example:



This style has a start and end, then a pattern of thin, medium and thick posts. The starts and ends also have connectors for the metal cables. You could break this design down into these 5 sections:



But actually if we look at the design a little more closely you'll see that even within those segments there's quite a lot of repetition.The cable connectors and the large posts are repeated in one of the evenly segments and also the Start and End segments. So we can break this down even further, like this:



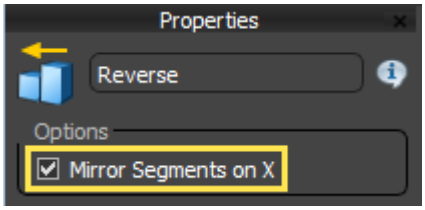
This version allows you to re-use the cable connectors and the large post for both the Evenly and the Start and End segments. It may seem counter-intuitive that in this version we now have an additional segment but if you have to edit the style, for example the cable connectors, you'll only need to do it once. In the previous version you would need to edit the cable connectors on the Start and End segment and both sides of the large Evenly post to make a change. Editing in multiple locations like this can have a negative impact on a fast, iterative work flow, which is why it is beneficial to break these models down as much as possible. To understand how to do this it helps to see how RailClone is used to reconstruct a style like this, so in the following exercise we combine these 6 segments using Compose, Reverse and Sequence operators to create the railing style.

Exercise: Combining segments using compose and reverse operators

1. Open *chapter_2_compose* from the downloads for this guide
2. Select *rc_bridge_railing* and open the Style Editor. The segments have already been added to this style.
3. First we'll create the Start and End. Create a new **Compose** operator and wire the segment called *railing_end* to the **First** input. Next we want a post so connect *railing_post* to the **second** input. Finally we need the cable connectors, wire *railing_connector* to the **third** input. These 3 segments are now combined to create the start segment. Wire the Compose operator to the Generator's **Start** input,

4. To reuse the Compose operator for the End input you use a **Reverse** operator. This will invert the order of the 3 segments and also give you the option to mirror the geometry on the X axis. To use it, create a new **Reverse** operator and connect the **Compose** node to its input. Next wire the **Reverse** operator to the Generator's **End** input.

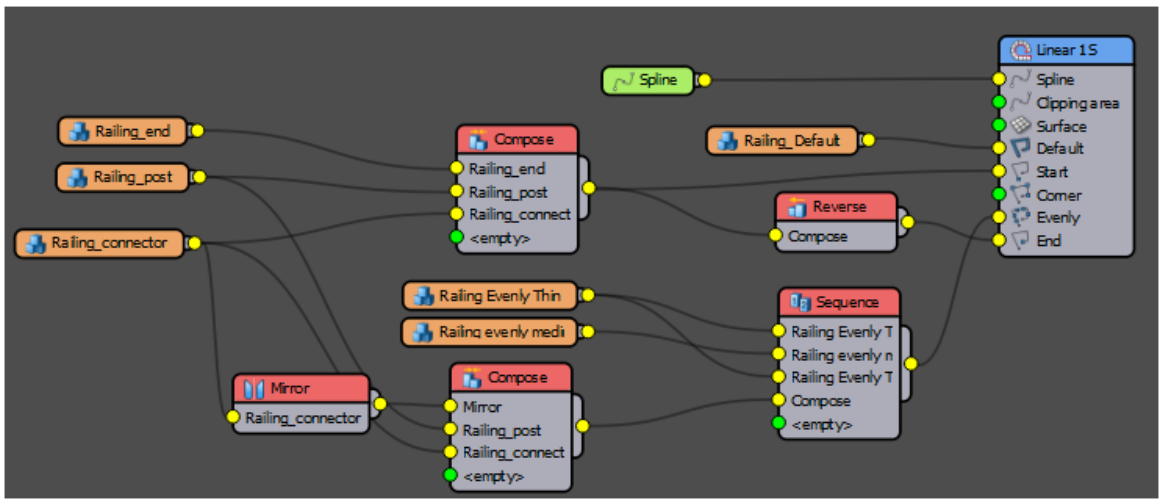
5. Select the **Reverse** operator and turn on **Mirror Segments on X.**



6. Next we'll create the large post with the cable connectors on either side. To do this create a new **Compose** operator and wire a **Mirror** operator to the **first** input. Wire *railing_connector* to the **Mirror** operator, and wire **railing_post** to the **Compose** operator's second input. Finally wire *railing_connector* without the mirror to the **third** input. This creates a large post with cable connectors on both sides.

7. As well as the large post, there are 2 other Evenly segments in this style: A thin and a medium post. Examining the finished design above you can see these are used in repeating pattern of thin post - medium post - thin post - large post. To achieve this, create a new **Sequence** operator. Wire the segments to the sequence operator in the correct order and connect the **Sequence** operator to the Generator's **Evenly** Input.

8. To complete the style, wire the *railing_default* segment to the Generator's **Default** input. The finished node tree is shown in the image below, notice how the *railing_post* and *railing_connector* segments are re-used in the **Start**, **End** and **Evenly** inputs.



Understanding how the compose operator combines segments to build new components can help you to build smaller and more efficient modular geometry.

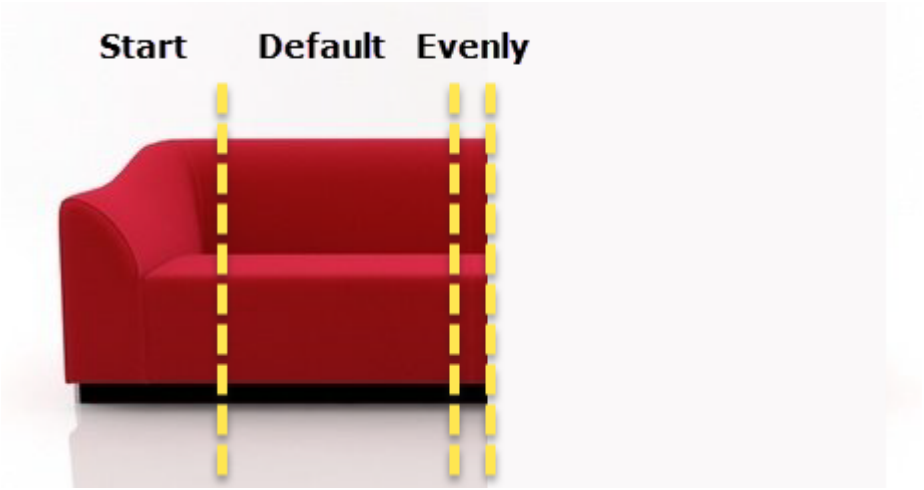
2.7 Rule 7 - Building components Individually vs using the Slice modifier?

Everyone has there own approach to modelling and it would be remiss of us to suggest the tools you should use to create geometry for RailClone. That said, there are broadly two possible methods you can use for modelling modular segments. You could either model each component individually, or model the full design as a single object and slice it into smaller parts.

Each has their advantages and disadvantages and your choice of which to use is often determined by the project at hand. If the object you are trying to create is already clearly divided in real-life into distinct pieces it is often easier to model it this way. Things like floors, ceilings, façade systems etc. are often made up of components like this, and this is often the best approach to model them for RailClone.

On the other hand if the model has elements which start in one segment and continue seamlessly to an adjacent segment, or if you've purchased or been provided with an existing model you'd like to parametrise, you'll probably find it easier to divide it into parts using a number of slice modifiers. For example, let's say that you wanted to parametrise this sofa from Design Connected.

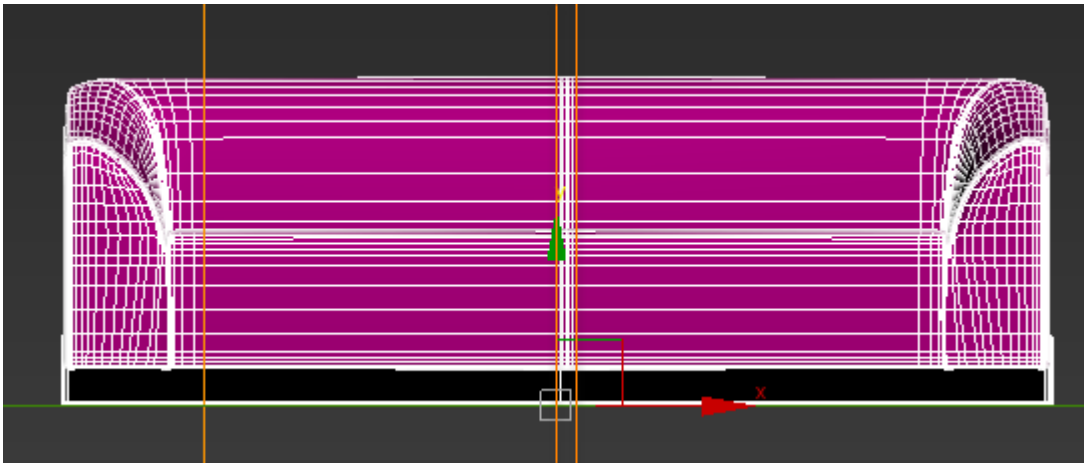
Firstly we need to think about how this will fit into a RailClone generator. In this case we only want to change the length so an L1S generator is most appropritate and we can split the sofa into these sections.



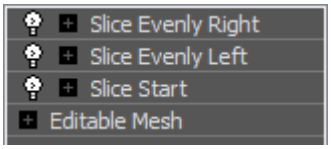
To do that you'd use 3 slice modifiers. Try the following exercise to see how they are placed.

Exercise. Slicing an object:

1. Download the sample model from Design Connected
2. Apply a **Slice** modifier and rotate the slice plane by 90 degrees. Move the slice plane on the X Axis so that it is just to the right of the sofa's arm. Rename this slice modifier *Slice Start*



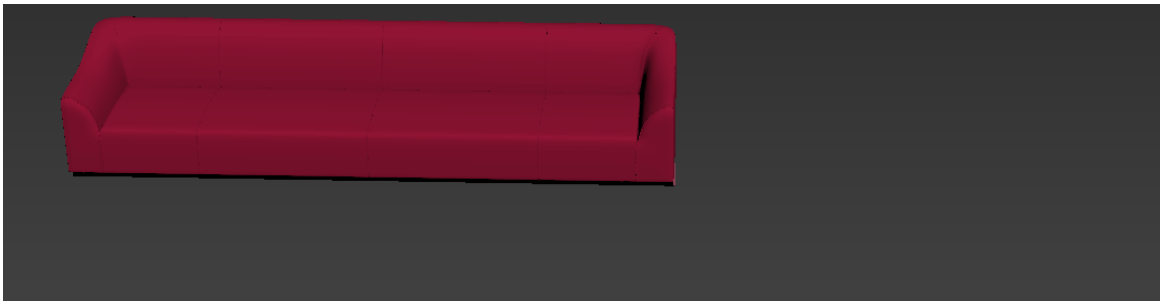
3. Right click on the slice modifier and select **Copy**. Right click again and click **Paste**. You now have a second slice modifier that can be moved to the left hand side of the seam in the middle of the sofa. Rename this modifier *Slice Evenly Left*.
4. Copy and Past *Slice Evenly Left* to create a third slice modifier and move its slice plane just to the right of the seam in the middle of the sofa. Rename this modifier *Slice Evenly Right*. You'll now have 3 slice modifiers that we can use to create the segments for an L1S array.



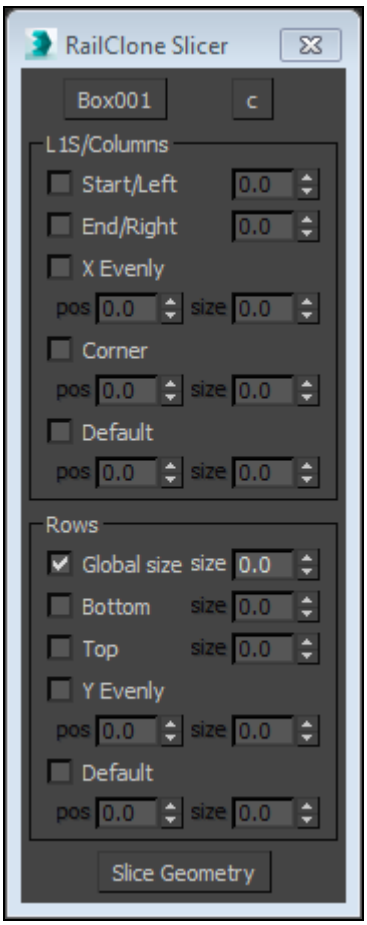
5. Select *Slice Start* and change the **Slice Type** to **Remove Top**. This will leave you only the arm of the chair. Rename the object *Sofa Start*
6. Duplicate the model by pressing CTRL + V. Change the *Slice Start* **Slice Type** to **Remove Bottom**. Select *Slice Evenly Left* and change the **Slice Type** to **Remove Top**. This will create the default segment. Rename the object *Sofa Default*.
7. Duplicate the model a third time by pressing CTRL + V. Change the *Slice Evenly Left* modifier's **Slice Type** to **Remove Bottom**. Select *Slice Evenly Right* and change the **Slice Type** to **Remove Top**. This will create the evenly segment. Rename the object *Sofa Evenly.*

To create the RailClone style for these segments, follow these steps:

1. Create a new Style with a single **L1S Generator** and a **Spline** input.
2. Create a new **Segment** node and pick *Sofa Start* as the source object. Change the segment's **Properties** > **Alignment** > **Y** to **Pivot**. Because all the objects were originally created from the original sofa, their pivots should line up perfectly.
3. Wire the new segment to the generator's **Start** input.
4. Create a new **Mirror** operator and wire the segment to its input. Wire the **Mirror** operator to the Generator's **End** input.
5. Create a new Segment, set the **Y Alignment** to **Pivot** and use *Sofa Default* as the source geometry.
6. Select the Generator and set **Properties** > **Rules** > **Default** > **Mode** to **Scale**.
7. Create a new Segment, set **Y Alignment** to **Pivot** and use *Sofa_Evenly* as the source geometry.
8. Wire the new segment to the Generator's **Evenly** input. You will now have a parametrised version of the sofa created by dividing the source geometry into segments using the Slice modifier.



Use a slicing script



To speed up the slicing process we use a script that allows you to add and controls slice modifiers and create segments that are ready to be used in RailClone's 1D or 2D Arrays. We've made it available here if case it's of use, but please note that this script is not supported officially by the iToo Software team , so please try at your own risk.

To use RailClone Slicer:

- Run the script and then pick the object from the scene.
- Activate the checkboxes for the rows and columns you need to create, and adjust the slice points using the position parameters (If you would rather position the slice planes manually, just pick and adjust them from the modifier stack, the segments should still be created correctly when you click **Slice Geometry**). When creating segments for an L1S generator you only need to use the settings in the **L1S/Columns** group. If you're using an A2S generator you'll need at least one option checked in each of the Columns and Rows groups. It's not possible to use the Rows group on its own.

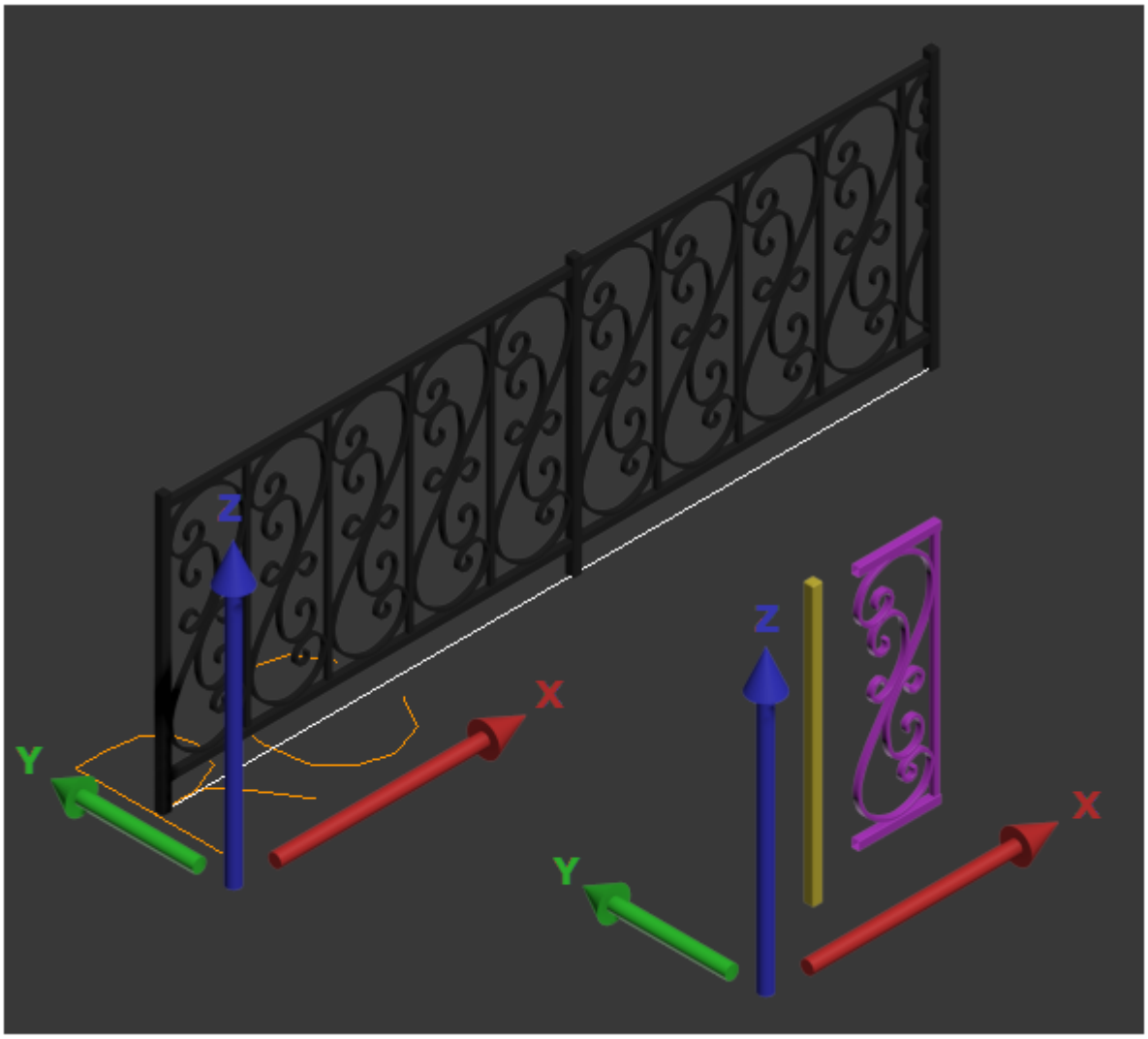
- To get an idea of what is being sliced, you'll need edged faces on, as it only shows one slice plane at the top of the modifier stack.
- When your cuts are in the right places, press **Slice Geometry** to create the segments
- Finally, if you'd like to remove the Slice modifiers from the original geometry, press the **C** button.

The script also aligns the pivots to world space by adding an XForm operator to the geometry when it is picked, so you need to ensure it is aligned correctly in relation to the world coordinates before use. Remember the L1S generator uses the X Axis to build the Array and A2S use the X and Y. Even if your final style will be oriented differently, for example in a vertical wall style the segment's pivots will still need to be oriented to these axes. You can read more about this in the next rule.

2.8 Rule 8 - Pivot orientation

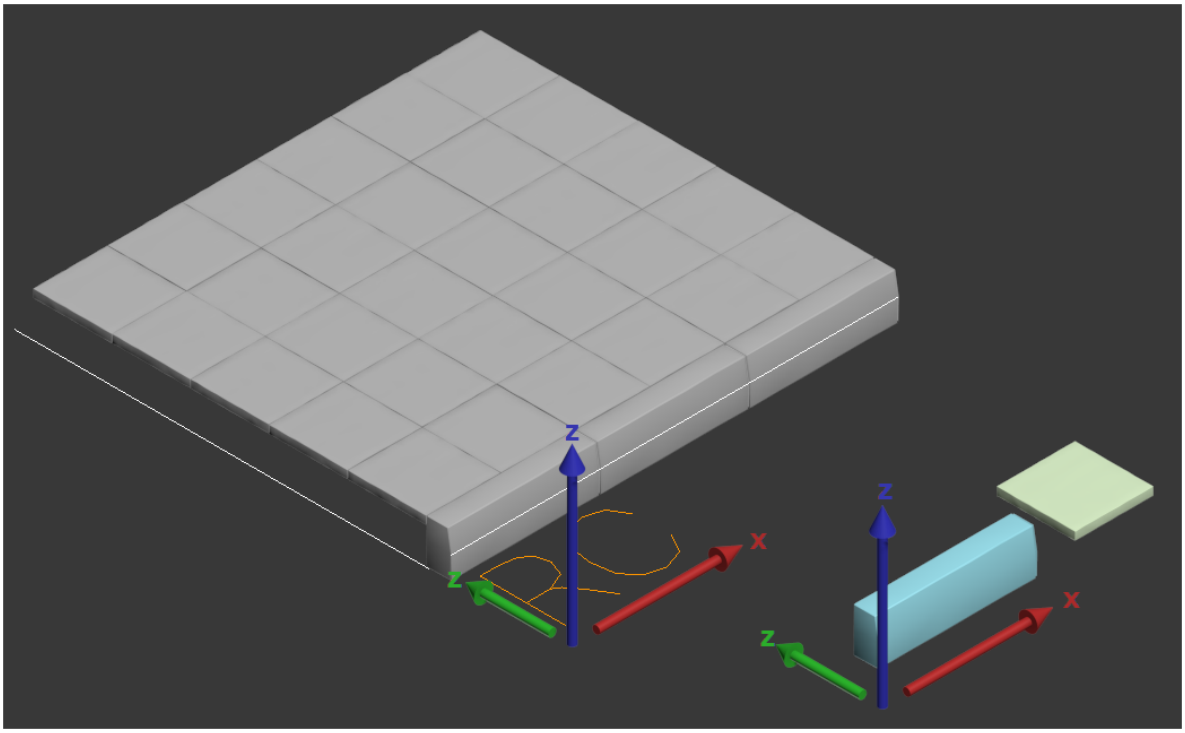
One of the problems new users often encounter is aligning segments correctly within the array. Often a segment is added only to find it is rotated incorrectly. The temptation at this point it to use the Segment's Fixed Transform settings to correct the rotation, but a lot of time and confusion can be saved by ensuring the pivots on the source geometry are oriented correctly. The L1S and A2S array both orientate the source geometry in the same way, but some confusion can arise when an A2S array is rotated to create a vertical style like a wall, window or facade. The 3 examples below should make clearer how RailClone uses aligns segments in the two array types.

Segment alignment in L1S Arrays:



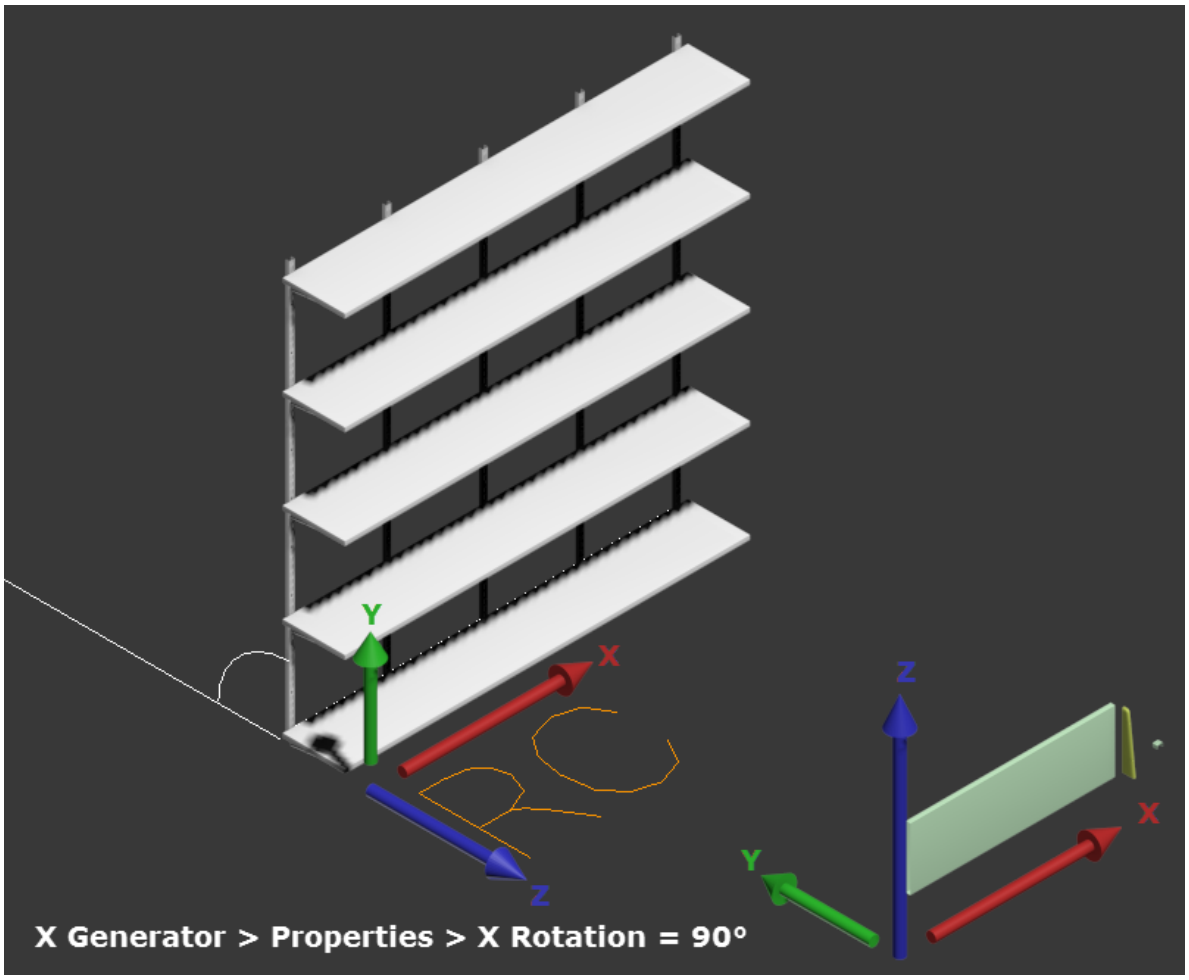
The local X axis of the source object is aligned to follow the spline. If no spline is used, the object is aligned to the RailClone object's local X axis.
The local Z axis of the source object is aligned to the RailClone object''s local Z axis.

Segment alignment in A2S Arrays:



The local X axis of the source object is aligned to follow the spline. If no spline is used, the object is aligned to the RailClone object's local X axis.
The local Y axis of the source object is aligned to follow the spline. If no spline is used, the object is aligned to the RailClone object's local Y axis.
The local Z axis of the source object is aligned to the RailClone object''s local Z axis.

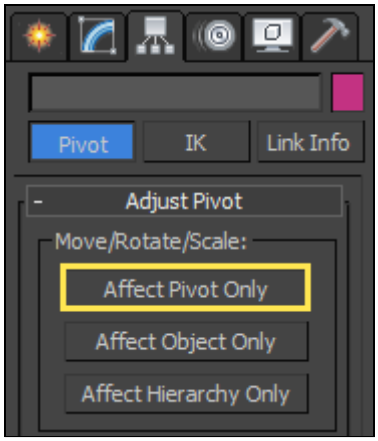
Segment alignment in Rotated A2S Arrays:



Some confusion can arise when an A2S array is rotated using the Generator's X Rotation feature. In this case the array is built as usual on the XY plan and then rotated around the X axis. In this case the final orientation of the source geometry and their final orientation in the array does not seem to match.

In order for geometry to be correctly aligned in the array, it may be necessary to adjust the orientation of the local axis. To do this:

1. Select the Geometry object.
2. Go to 3ds Max's **Hierarchy Panel**.
3. Click **Affect Pivot Only.**



4. Rotate the Gizmo to the correct orientation.

2.9 Rule 9 - Pivot placement

When you add a segment to RailClone, it is positioned in relation to the base spline and adjacent segments using the extents of its own bounding box. In order to accurately build a style it helpful to understand how this works, because the alignment varies **depending on where the segment appears in the array**.

Unless you specify otherwise, segments use an **Automatic** mode that changes the pivot used for placement according to whereabouts in the array the segment is used. For example, a segment used in the Start input of an L1S array will be aligned so that the left hand side of the bounding box is aligned with the first vertex in the array. The exact same segment if used in the End input will use the opposite side - the right of the bounding box. And if you use it in the Evenly input, it will use the centre of the bounding box.

Though this seems confusing it is in fact quite logical when you consider why these changes are taking place. For the start segment to use anything but the Left of the bounding box as a pivot would result in a mismatch between the length of the path and the length of the geometry.

In the table below you can see the alignment used when a segment is in Automatic mode wired to the different inputs of an L1S array. With an L1S array the Y and Z alignment remain constant, only the X alignment changes.

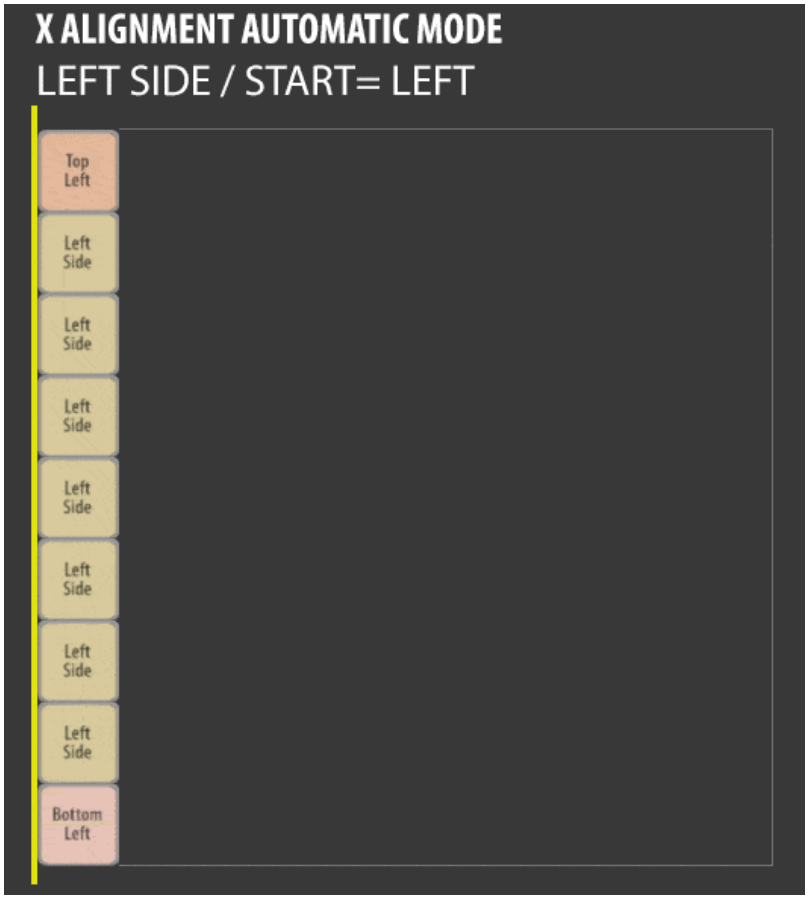
Start	Default	X-Evenly	Corner	End
X:Left | Y: Centre | Z: Bottom	X:Left | Y: Centre | Z: Bottom	X:Centre | Y: Centre | Z: Bottom	X:Centre | Y: Centre | Z: Bottom	X:Right | Y: Centre | Z: Bottom

This is different in an A2S array where only the Z alignment is constant, The X alignment is the same as the L1S array and the Y alignment now varies depending on which Row a segment is used in.

	Left/Start	**Default**	**X-Evenly**	**Corner**	**Right/End**
Top	X:Left | Y:Top | Z: Bottom	X:Left | Y:Top | Z: Bottom	X:Centre | Y: Top | Z: Bottom	X:Centre | Y: Top | Z: Bottom	X:Right | Y:Top | Z: Bottom
Y - Evenly	X:Left | Y: Centre | Z: Bottom	X:Left | Y: Centre | Z: Bottom	X:Centre | Y: Centre | Z: Bottom	X:Centre | Y: Centre | Z: Bottom	X:Right | Y: Centre | Z: Bottom
Default	X:Left | Y: Bottom | Z: Bottom	X:Left | Y: Bottom | Z: Bottom	X:Centre | Y: Bottom | Z: Bottom	X:Centre | Y: Bottom | Z: Bottom	X:Right | Y: Bottom | Z: Bottom
Bottom	X:Left | Y: Bottom | Z: Bottom	X:Left | Y: Bottom | Z: Bottom	X:Centre | Y: Bottom | Z: Bottom	X:Centre | Y: Bottom | Z: Bottom	X:Right | Y: Bottom | Z: Bottom

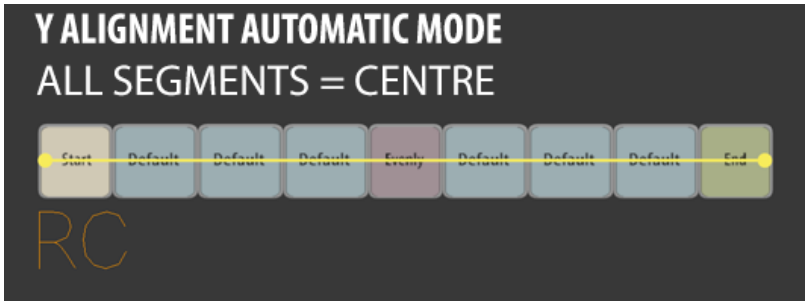
The images below give you a visual representation of these alignment changes in action. As you can see they are necessary to ensure that the geometry fits within the limits of the size of the array.

2.10 X Alignment



L1S and A2S

2.11 Y Alignment

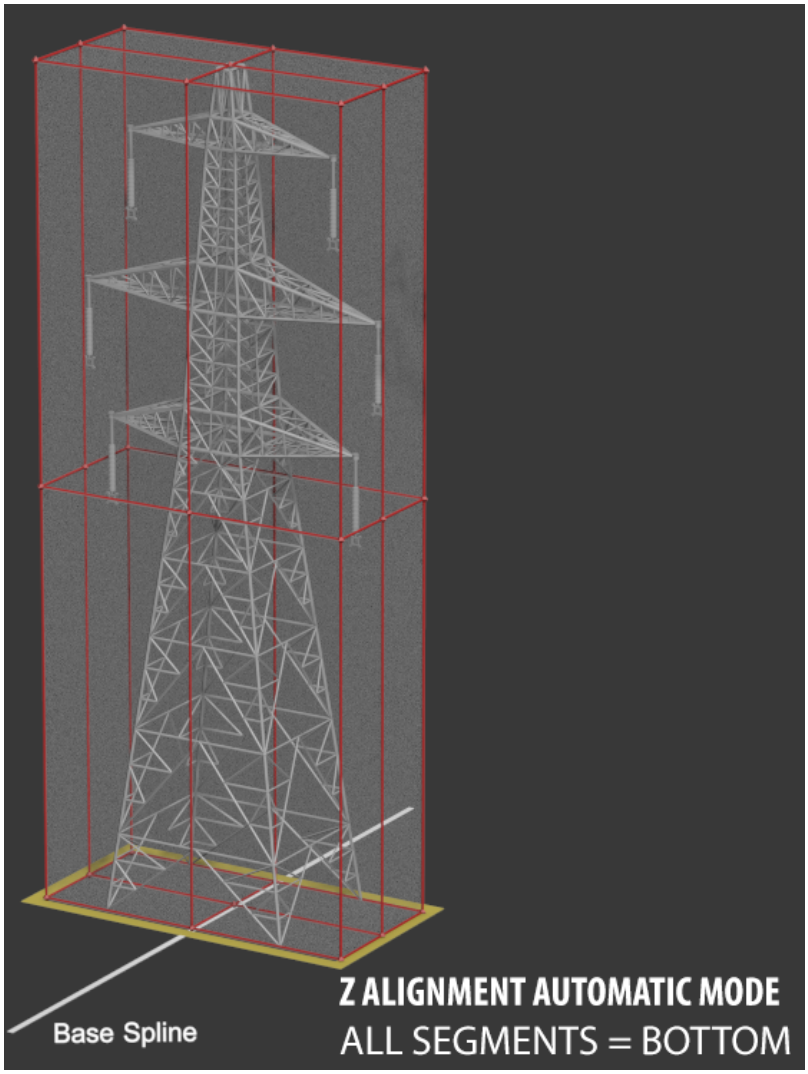


L1S

Y ALIGNMENT AUTOMATIC MODE
BOTTOM SIDE = BOTTOM
Bottom Left
Bottom Side
Bottom Side
Bottom Side
X Evenly
Bottom Side
Bottom Side
Bottom Side
Bottom Right

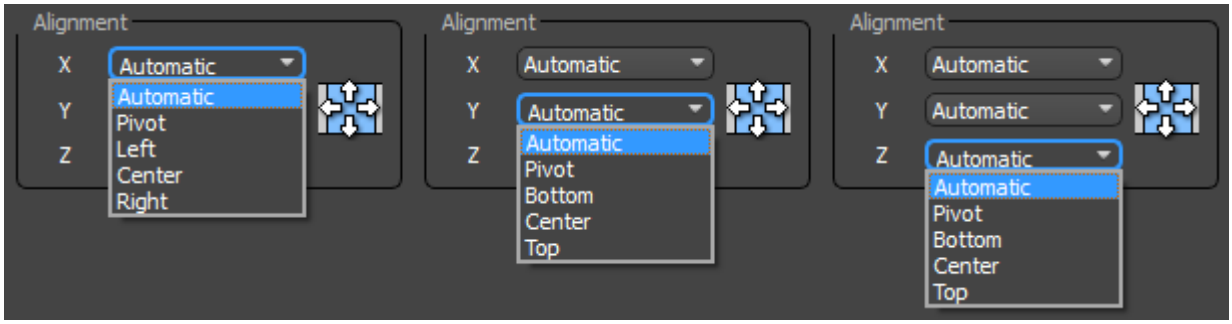
A2S

2.12 Z Alignment

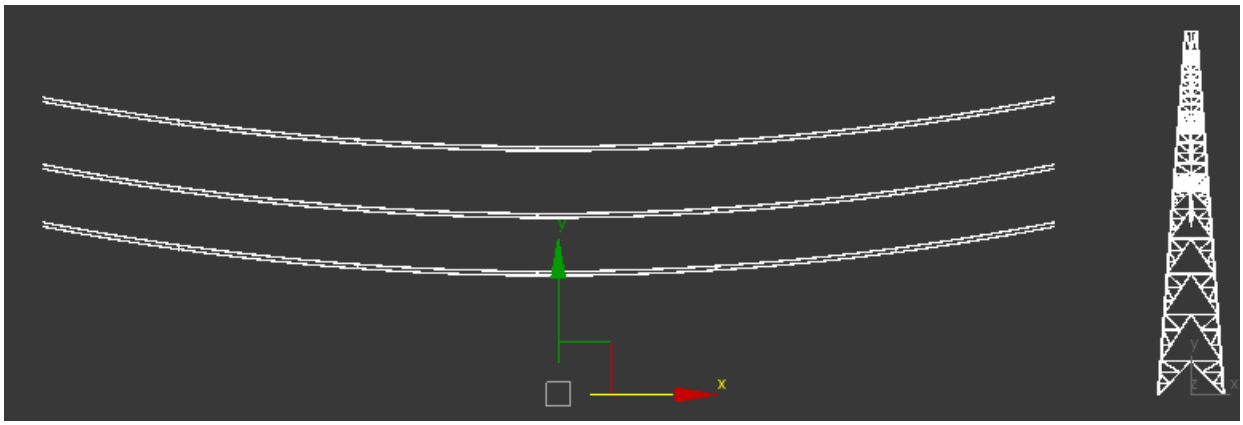


L1S and A2S

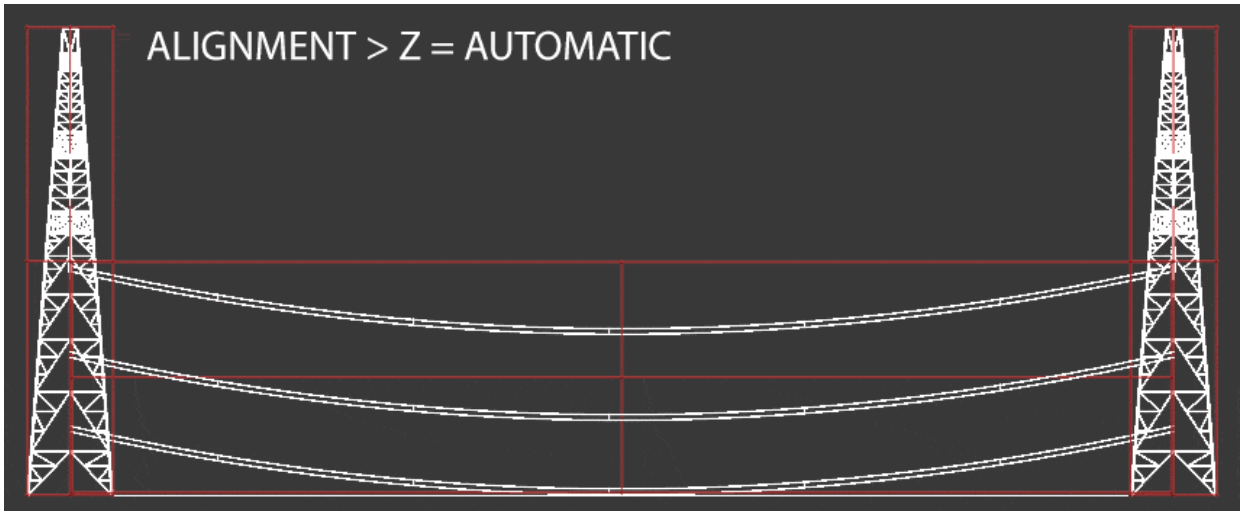
In most cases Automatic alignment works well, but in some scenarios you may want to manually specify how the segment is aligned. For each axis you RailClone offers a number of different options as shown in the below.



For example, a simple Pylon style with a segment for cables and another for the towers will not work well with default pivot alignments. The side view below shows these segments:



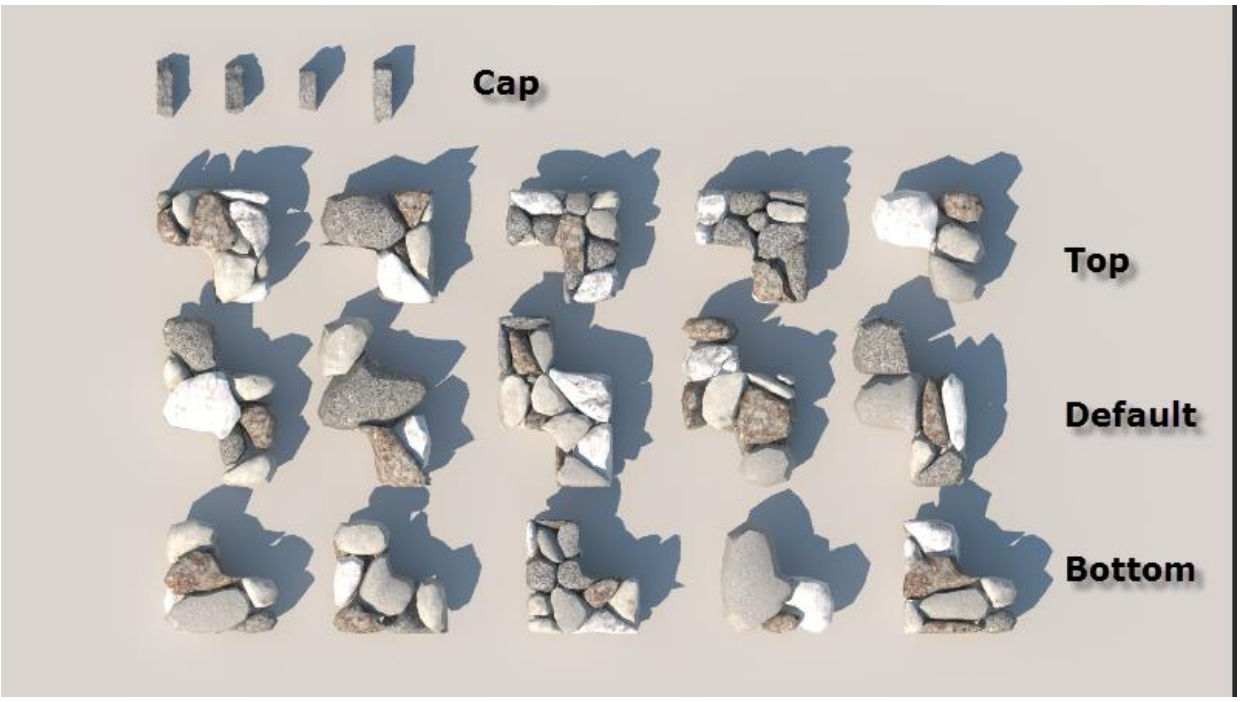
The automatic alignment would move the cables down to the ground so that the bottom of their bounding box is on the spline. We could of course move it back into position in RailClone using the Segment's fixed transform values, but by setting the pivot correctly on the source geometry and changing the Segment's **Properties** > **Alignment** > **Z** value from Automatic to **Pivot** the problem is solved much more easily.



To see this style, download *chapter_2_pylons.max*.

2.13 Rule 10 - Disguise joints with interlocking geometry

RailClone is grid based which can occasionally be a problem when you want to create the illusion of a randomised pattern. To get around this you can create an illusion of chaos by creating overlapping segments of the same size and shape but that contain randomly shaped elements. For example, in this example of how to approach a dry stone wall you can see tetris-like individual segments that are consistent in size and shape, however within each one there is no clear order.



When these lock together the grid of the underlying array becomes much harder to detect.

2.14 Rule 11 - Reset XForms

Generally speaking it's a bad idea to scale geometry at object level, but if it is unavoidable or you've rotated the object and you want a quick way to reset the pivots to world space, then use the Reset XForm tool found in Max's utilities menu. If you don't then these changes will not be reflected when the segment is imported into RailClone.

To reset the XForm:

1. Select an object.
2. Go to the Utilities panel and click Reset XForm.
3. From the **Reset Transform** rollout, click **Reset Selected**.

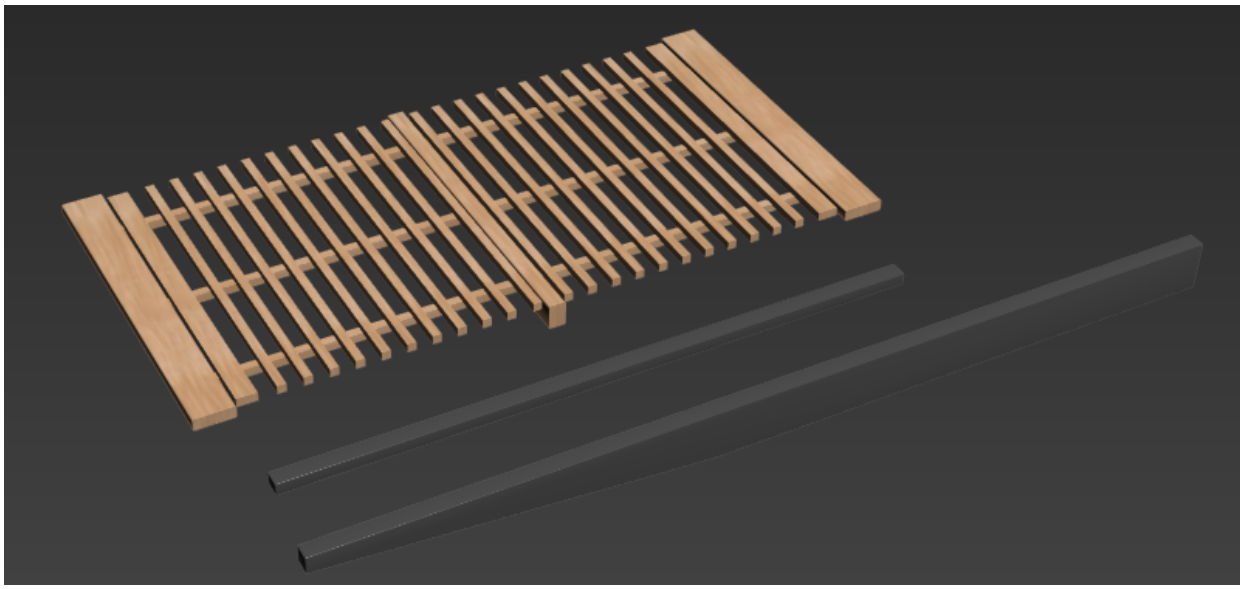
> As well as resetting scale values, the Reset Xform utility realigns the pivot to world space, so you'll want to make sure that the object is orientated correctly before using it.

The XForm modifier can also have another useful purpose. If you want to animate the transforms of the source geometry, you can apply a XForm modifier to the object, select the Gizmo sub object and animate the rotation. translation or scale. Unlike animating the object directly, keyframing the Gizmo allows the animation to be maintained when the geometry is added to RailClone.

2.15 12 - Correctly assign material IDs to “parts”

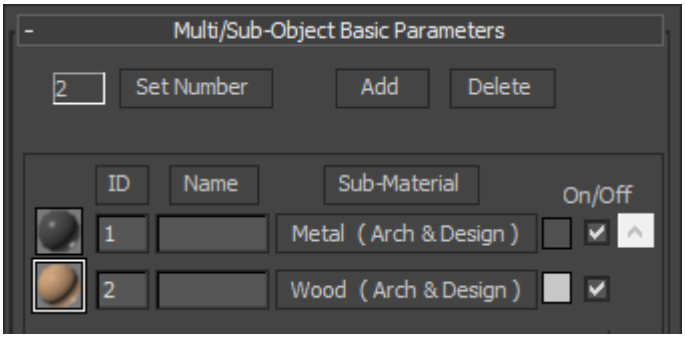
When creating custom geometry for RailClone, it's important to take into consideration the management of the object's material IDs. If you've used Forest Pack before, there may be some initial confusion because RailClone does not automatically consolidate materials and reassign IDs automatically. At present you need to manage this manually by creating Multi/Sub Object materials that use material IDs set manually on the source geometry. In effect the same material applied to both RailClone object and the individual components will yield the same results.

Take a simple brise-soleil for example. The two segment's shown below both use a separate material: a metal for the support and a wood for the shades. Each is currently a separate material, and the material ID assigned to the faces of both objects is set to 1.

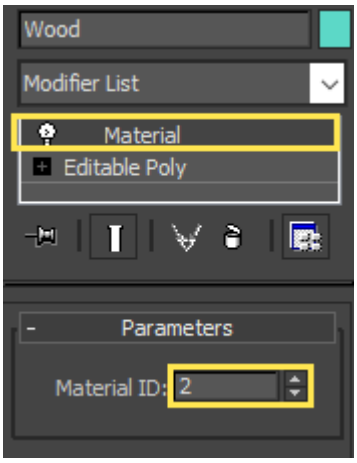


When you add these segments to RailClone, you'll notice that only the wire colour is used. This is because a material needs to be assigned to RailClone manually, but here's the problem. You have two materials on the source geometry but only one of these can be applied to RailClone, so it's necessary to a build a multi/sub object material that works when applied to all the individual segments. This will almost certainly result in needing to edit the material IDs. To illustrate this, let's fix this style.

1. Open *chapter_2_brise_soleil* from the the downloads for this guide.
2. Open the material editor and create a new **Multi/Sub Object material w**ith 2 IDs. In slot 1 add the metal material, and in slot 2 add the wood material.



3. Assign the Multi/Sub-object material to the brise-soleil parts and the RailClone object. You'll notice that because everything has an ID of 1 it uses the metal material.
4. Select the wooden part of the brise soleil and add a **Material** modifier.
5. Change the Material ID value in the material modifier to 2 to match the Multi/Sub-Object material we just created.



6. The RailClone object updates automatically and any geometry with an ID of 2 now uses the wood material.

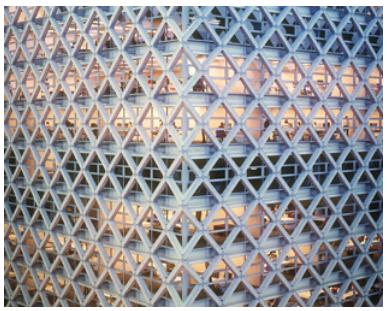
Tips to automate building multi-sub object materials

If you don't want to create Multi/Sub-object materials and re-assign IDs manually, an easy way to consolidate materials is as follows:

- Attach all the objects into a single mesh. Use the default options, so Max creates the multi/sub material automatically.
- Detach each object back into separate objects.
- Re-adjust the pivots if necessary.

> You may also find there are some scripts available that create a new Multi-Sub object material and reassign IDs for you. For example we use RappaTools as it has a **Create Multi-Material from Selection** option. We believe the free version may have the same functionality, but we haven't tried it extensively. Another option is the Material Combiner script by Brian Keffer which performs a similar function.

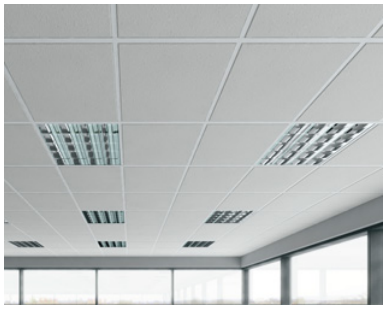
2.16 Related Tutorials



populate_building_interiors

Populate Building Interiors

The Populate Building interiors tutorial demonstrate how RailClone and Forest pack can be used together and also features section on creating a lattice facade that is a useful demonstration of how to create a style from minimal components using Sequence and Mirror operators.



creating_ceilings

Creating Ceilings

This tutorial explains how to create a ceiling. The parts are all modelled separately based on profiles supplied by ceiling manufacturers.

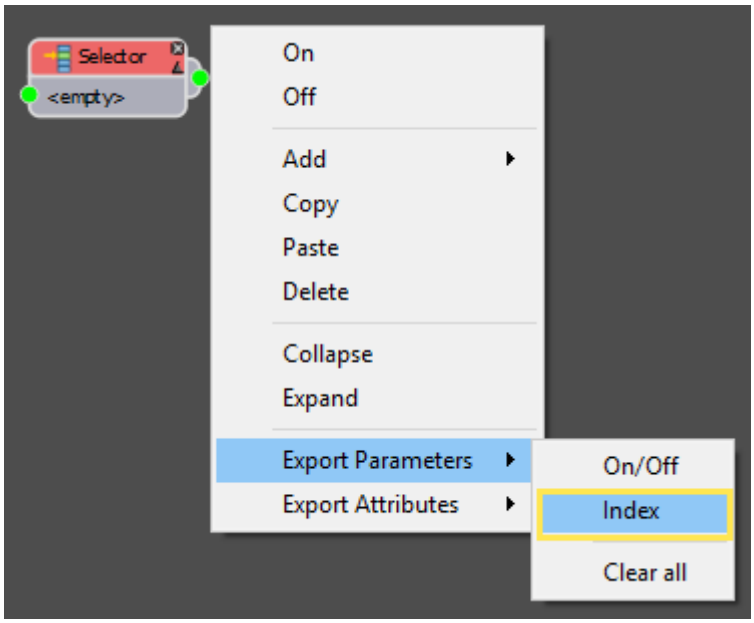
For more information please see the 1D arrays - Generator L1S, 2D arrays - Generator A2S, Segments and Rendering Best Practices sections of the online documentation.

3 3 - Wiring Parameters and Attributes

Almost every value in every object, operator, generator, and parameter node can be **Exported** so that it has its own input. This is a powerful feature that allows you to control a node's properties from the modify panel to make sharing and reusing your styles a lot easier , or you can build numerical relationships between parts of the graph using exported attributes and arithmetic operations.

3.1 Exporting Parameters

Exporting Parameters is very simple. Right click on any node and at the bottom of the menu is an option to **Export Parameters** that contains a list of all the values that can be exposed to create a new input.



Just select the property and new input will be created at the bottom of the node.

Once you've exposed a property it can be wired to 4 types of other node:

1. Constant Nodes

Constant nodes allow you to enter a value that can only be changed from within the style editor. Unlike the Numeric node, this property cannot be animated.

2. Numeric Nodes

Numeric nodes are used to create values that can be edited outside the style editor in the Modify panel. They are commonly used to create a simplified interface so that the user can interact with the style without needing to open or understand the Style Editor. Unlike the Constant node, this value can be animated.

3. Exported Attributes

The current size of a segment's bounding box can be exported from virtually any node. The process is very similar to exporting a parameter, just right click on a node, select **Export Attribute** and choose the parameter to be exported. The new exposed Attributes will be added to the bottom of the list of output slots.

4. Arithmetic Nodes

Arithmetic nodes are used to perform common mathematical operations on values from attached numeric nodes, constant nodes , and attributes. Using expressions it is also possible to access a number of advanced variables and functions.

Exercises

In this chapter, to illustrate how expressions work we'll make a tile pattern from a single box object. Using expressions we can include the following features, all of which will adjustable from the modify panel:

- Create a pattern with alternating rows and columns of half sized tiles. The size of the small tiles are automatically calculated based on the the dimensions of the large tile.
- Ability to adjust the height width and length of the large tiles from the modify panel.
- Ability to rotate the style from the modify panel.
- Ability to toggle on/off or off the rows and columns of half size patterns.

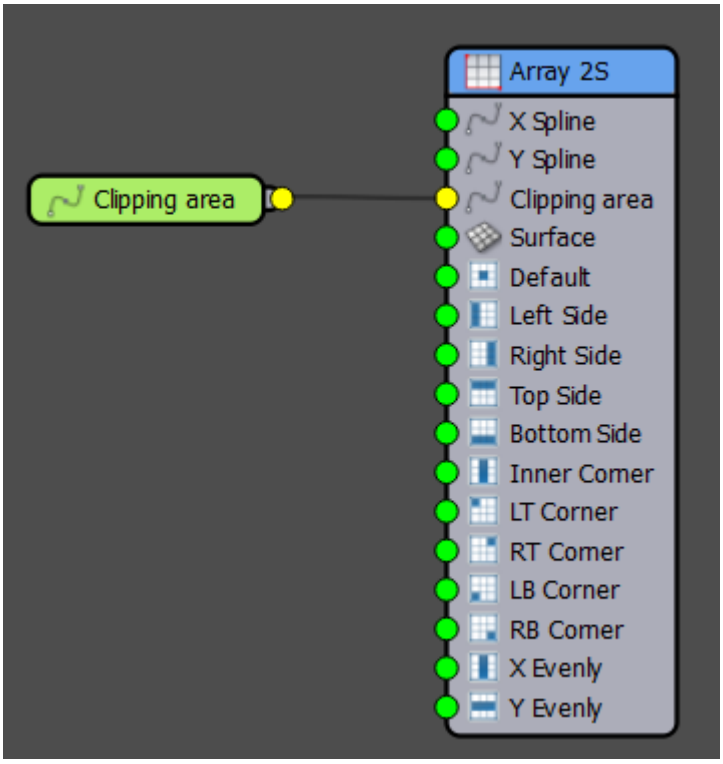
Please see the video below for a preview of this style.

Exercise: Setting up the style.

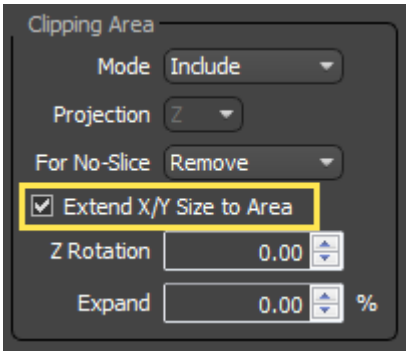
Before we start exporting and wiring parameters let's create the basic style. In the scene file for this chapter only a single box and an empty RailClone Style is supplied. Add the large tiles and create the initial array by following these steps.

1. Open *chapter_3_expressions_start.max* from the downloads for this guide.

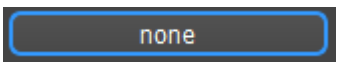
2. Select the RailClone object and open the Style Editor.
3. Create a new 2D array by dragging an **A2S** generator from the Items List to the Construction View.
4. Create a new Spline node and wire this to the Generator's **Clipping Spline** input.



5. The clipping spline can be used to automatically set the size of the array, to do this select the generator and turn on **Extend X/Y Size** to area from the properties.



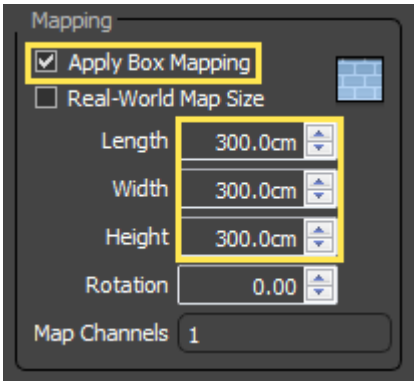
This feature is discussed in more detail in Chapter 6 of this guide.

6. Select the Spline node, click on 

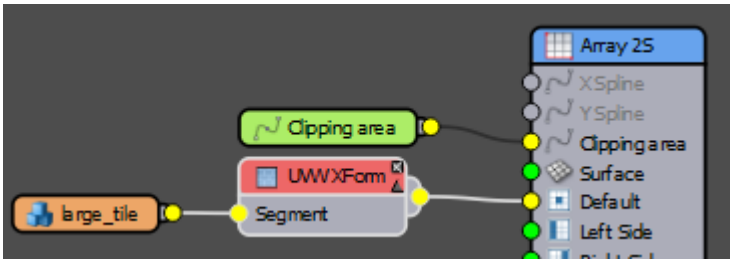
from the Properties panel and pick *boundary_spline* from the scene. The array will be created so that it fills the area represented by this spline.
7. To add the tile geometry, create a new **Segment** node, go to the properties panel and click

then pick *large_tile* from the scene.

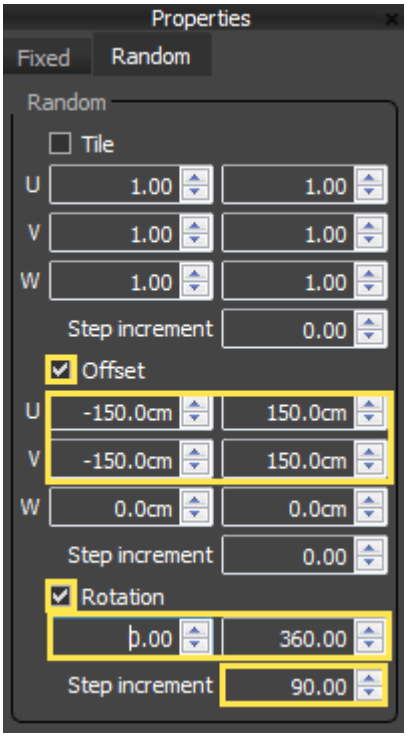
8. Because we're going to rescale this segment and we don't want the textures to become distorted we can automatically box map the geometry directly in RailClone. To do this, select the segment node and go to the **Deform** tab of the **Properties** panel. Turn on **Apply Box Mapping** and enter a value of **300cm** for the **Length**, **Width** and **Height**.



9. You will now have a simple tile style. Notice though that though we have mapped the geometry, the texture on each tile is identical. To add some variation we can randomise the uv mapping coordinates. To do this add a **UVW XForm** operator and wire it between the segment and the generator.



10. Select the UVW XForm operator and open the **Random** tab in the **Properties** panel. Turn on **Offset** and enter a value of **-150cm** for **U** and **V Min** and **150cm** in the **U and V Max**. Also turn on **Rotation** and enter a value of **360** for the **Max** with a step value of **90** .

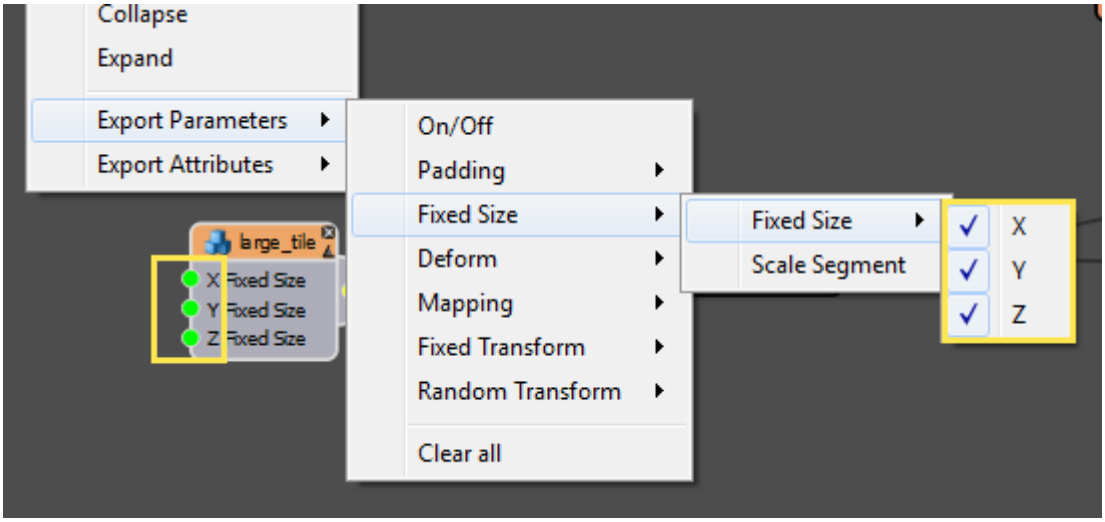


> We'll look at the UVW Mapping and UVW Xform options in more detail in the next chapter.

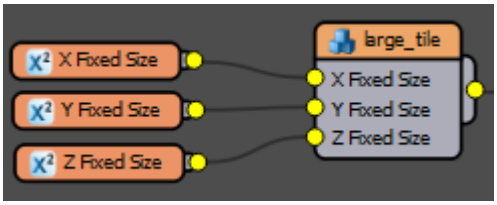
Exercise: Exporting Parameters and editing them from the Modify Panel

In this exercise we'll add some controls so the size of the tile and rotation of the entire array can be controlled from the modify panel. To achieve both these tasks it is necessary to export the parameters and attach them to a **Numeric** node. This is one of two types of node in which you can enter an integer, float, scene units, percentage or boolean value. The other type is called a Constant and differs only in that you cannot adjust it from the Modify panel. Generally speaking you would want to use Numeric Nodes where the value is likely to be changed regularly and Constant nodes when the value should remain unchanged.

1. Select the Segment node and Right Click. The Export Parameters options are at the bottom of the Right-Click Menu. Choose **Fixed Size** > **Fixed Size** > **X** and a new input appears on the Segment node. Do the same thing to export the **Y** and **Z** fixed size parameters.

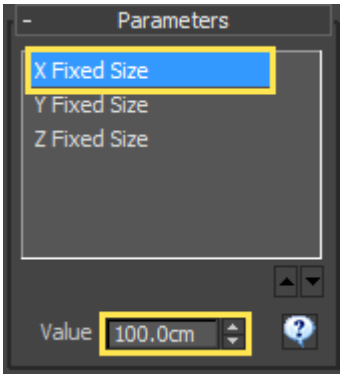


2. Create 3 new Numeric nodes, one for each exported parameter and wire them to the size inputs. You'll notice that the names of the Numeric nodes change automatically to reflect the input's they're wired to but if you want to name them manually you can do so from the **Properties** panel.

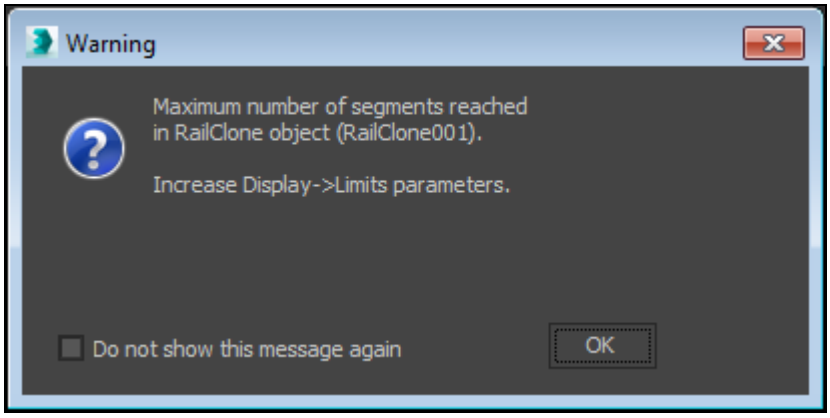


3. For each Numeric node, go to the **Properties** panel and change the type to **Scene Units**. You can also set the numeric type to float, integer, percentage or boolean but for dimensions scene units is the logical choice.

4. RailClone's **Modify** panel interface has a rollout called **Parameters**. This contains editable values for any **Numeric** nodes found in the style. In the **Parameters List** you should see the 3 Size values we created in the previous step. If you edit these values, the geometry in the style is resized!

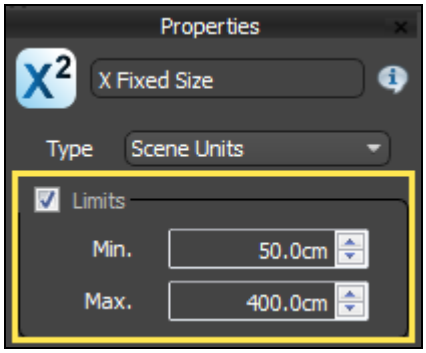


If you enter a very small value in the X or Y fixed size you may get a warning about exceeding the segment counts.



This happens because the small geometry size has caused too many segments to be created. If you increase the size the object should display correctly.

If you wish to avoid this issue you can set a minimum and maximum limit for the value by selecting the **Numeric** node in the Style Editor and turning on **Limits**.



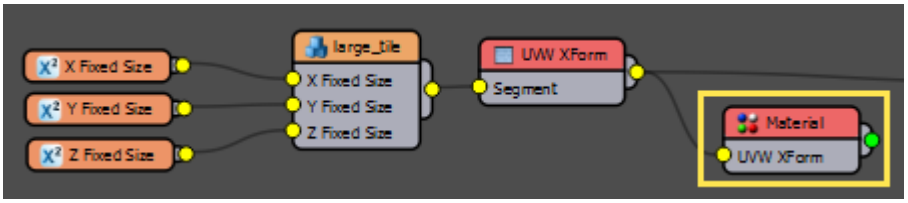
> However in this example I chose not to do this because when the Fixed Size parameter uses a value of 0, the dimensions of the original geometry are used instead. This gives me the option to set a size manually by entering a value, or use the original dimensions by leaving them set to 0.

5. It is also possible to export parameters from Generators. In this example we'll export the Z rotation property so it can be adjusted from the Modify Panel. Right click on the **Generator** and go to **Export Parameters** > **Clipping** > **Z Rotation**.
6. Create a new **Numeric** node and wire it to the **Z Rotation input.**
7. From the Properties Panel, change the Numeric node's **Type** to **Float**. Float makes sense in this case as the rotation value is measured in degrees.
8. In the Modify Panel, you will now have a new Z Rotation parameter. Changing this value rotates the entire array within the clipping spline.

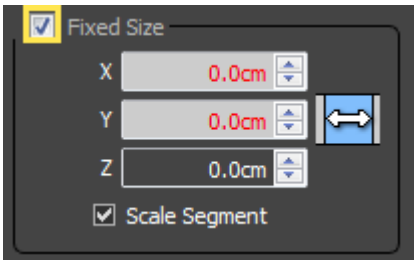
Exercise: Exporting Attributes and simple maths with the Arithmetic operator

Now that we have the main tile finished we can add the alternating rows of half sized tiles.

1. Firstly, change the material for the smaller tile. To do this, add a new **Material** operator and wire the output from the **UVW XForm** operator to the Material node's input.
2. Select the **Material** node and from the properties panel, change the **From** and **To** value to **2**. There is already a multi-sub object material applied that has a grey texture in ID slot 2.



3. Add a new **Transform** operator and wire the Material Node to its input. Transform operators allow you to re-use geometry by changing the padding, alignment, fixed size and transforms without affecting the original. In this example we will use the transform operator to scale the tile to half the size, so open the Properties and turn on the **Fixed Size** override toggle button.

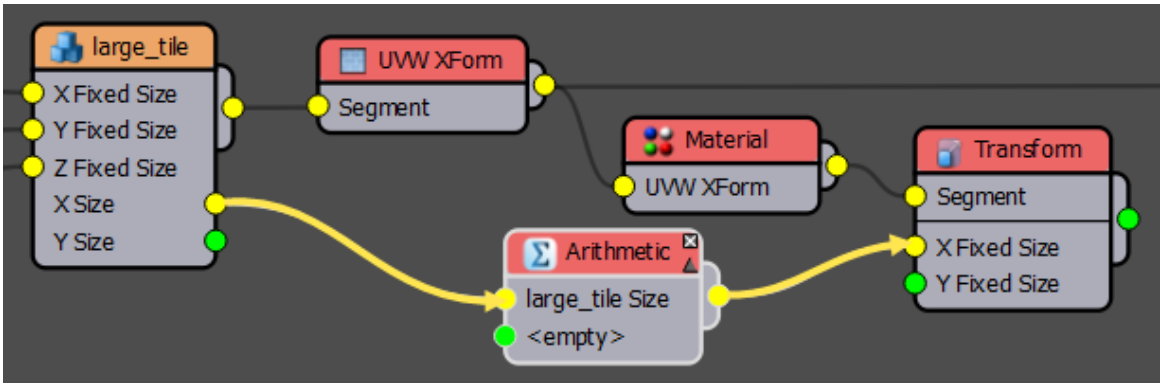


4. To be able to control the Fixed Size we'll export the X and Y size parameters. Right click on the Transform node and click **Export Parameters** > **Fixed Size** > **Fixed Size** > **X** and **Y**.

5. Next we need to get the existing size of the large tile. To do this you can export **Attributes** from any node. These will give you the current X Y and Z dimensions from any node at any point in the graph. Right click on the Segment node and select **Export Attributes** > **Size** > **X** and **Y**

6. To calculate the new size of the small tiles, create a new **Arithmetic** node and wire the **Segment's X** size output to the Arithmetic node's first input.

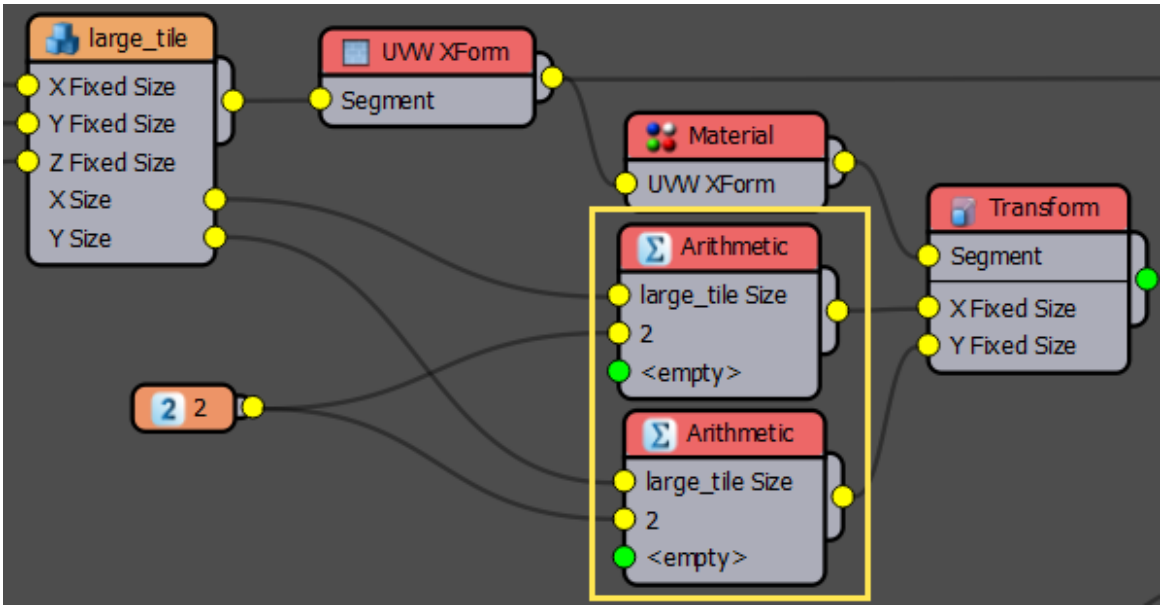
7. Wire the **Arithmetic** node's output to the **Transform** nodes **X Fixed Size** Input.



8. To divide the value in half, add a **Constant** node and enter a value of **2** in the Properties panel. Wire this to the **Arithmetic** node's second input.

9. Select the **Arithmetic** node. If you go to the properties panel you'll see it has many different operations including add, subtract, divide, maximum and minimum. In this case set the operation to **Divide**.

10. To calculate the Y size of the small tile we just need to repeat this process. create a new **Arithmetic** node and wire the **Segment's Y size** output to the second Arithmetic node's first input. Wire the **Arithmetic** node's output to the **Transform** nodes **Y Fixed Size Input**. Wire the existing constant to its second input and set the mode to **Divide**. This will now generate a half sized segment on both axes.



11. To see this in the style, wire a **Sequence** operator to the **Generator's Default** input. Wire the **UVW Xform** node to the first input, and wire the **Transform** operator to the second input.

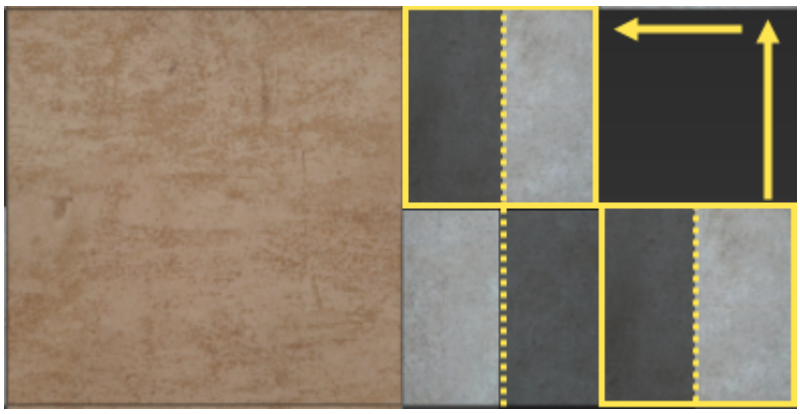
Congratulations! you have now exported values to control the size of tiles from the modify panel and used attributes and arithmetic nodes to automatically generate tiles that are always half the size of the main tile. In the next exercise we'll move the tiles into the correct position using padding and fixed transform nodes with a simple expression.

Exercise: Using Expressions

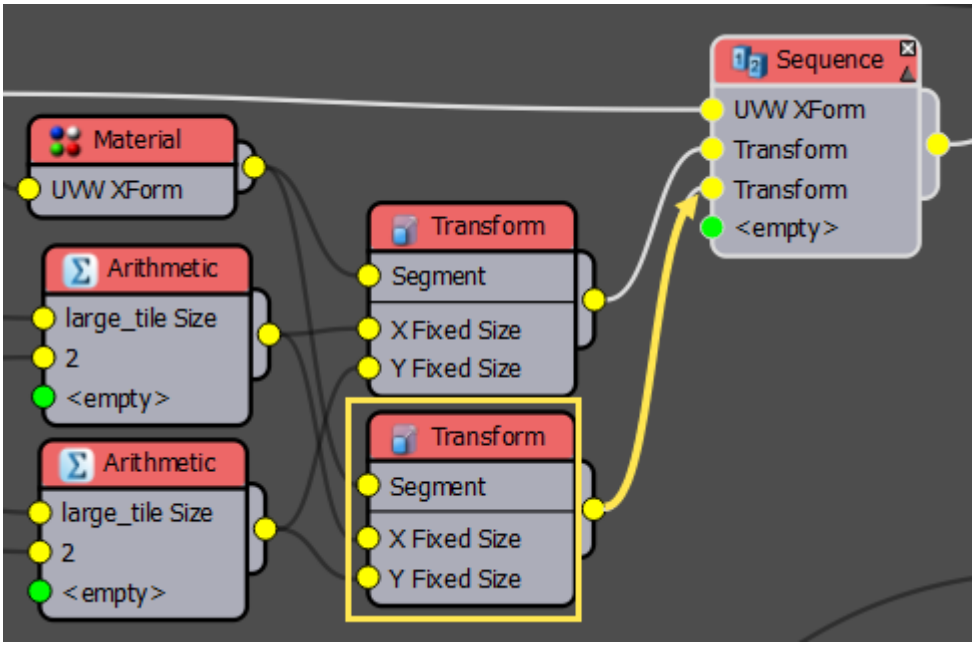
If you have completed the previous exercise correctly, there should be a repeating pattern on the X axis of a large tile followed by a half size tile. Next we need to add a second small tile in the gap marked in yellow in the diagram below:

To do this we will need to add another small tile to the sequence, and then move it **left** and **up** into the correct position.

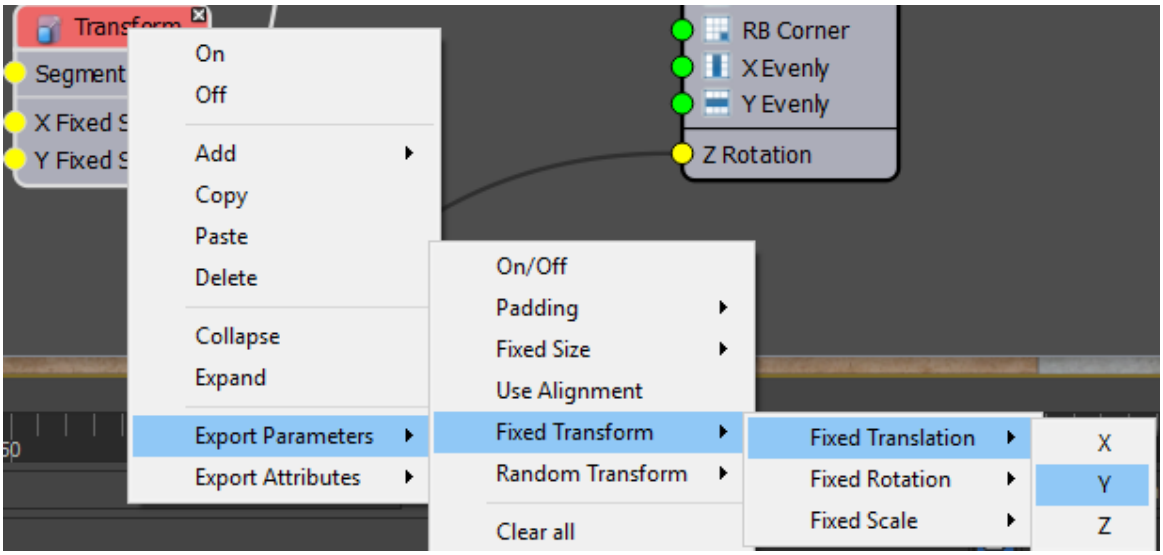
Moving the tile up is easy enough, we can use the **Y Fixed Translate** property and the existing Arithmetic node that's being used to calculate the tile's Y Size. However to move it to the left we can't use X Fixed Translate because it will leave a gap in the array. Instead we need to use Padding. Here thought there is a slight problem, in order to align correctly, the padding value needs to be the same as the width of the tile which will result in a segment that has no measurement on the X axis (tile width - tile width = 0). A segment with a size of 0 is disregarded in RailClone, as it can potentially result in the creation of an infinite number of segments. To get around this we deduct half of the size of the small tile from the right of the first segment, and half from the left of the second small segment. In the image below the padding on the tiles is marked in dark grey, as you can see when you disregard these areas the second tile slides to the left and into the correct position.



1. To create the padding we will need to duplicate the existing Transform node as the two small tiles need to have different settings. To do this, select the **Transform** node, right click and then select **Copy**. Right click again and select **Paste**.
2. Wire the new **Transform** operator to the **Sequence** operator's **third** input.

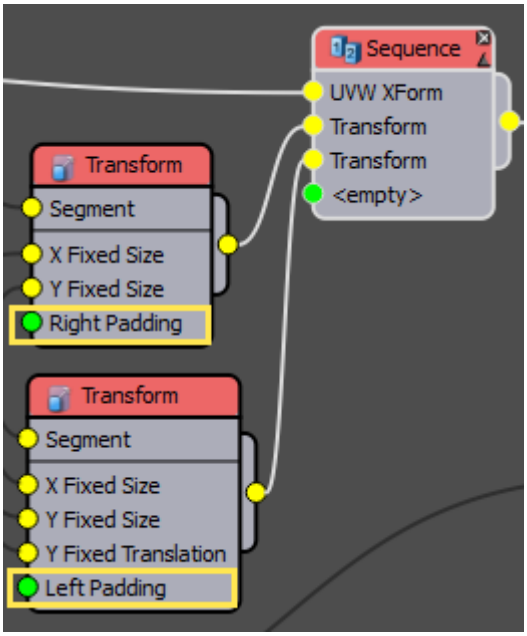


3. To move the new segment up, Right click on its Transform node and select **Export Parameters > Fixed Transform > Fixed Translation > Y**.

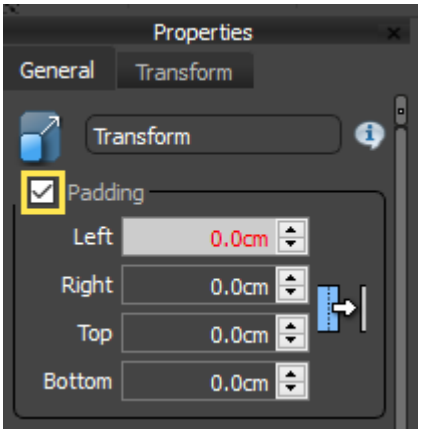


4. Wire the existing Arithmetic node that is used to calculate the height of the small tile to the **Y Fixed Translation Input.** The tile will now move up the height of one tile.

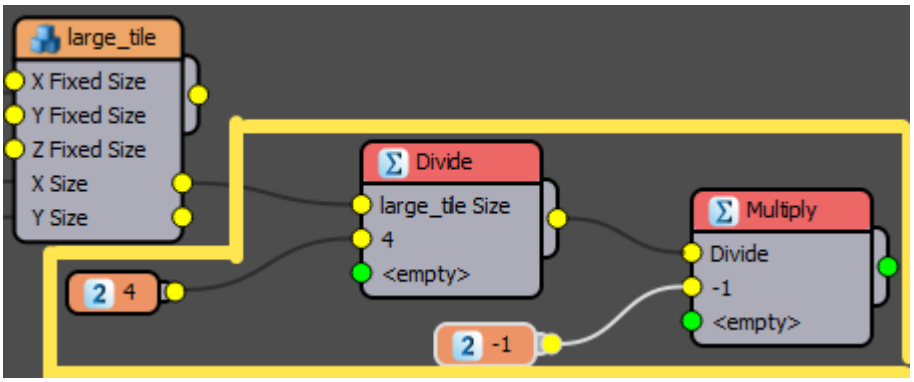
5. To add the Padding, export the **Padding** > **Right Padding** value for the first small tile and **Padding** > **Left Padding** for the second small tile



6. Turn on Padding in both Transform nodes. **Note that exported values are shown in red.**



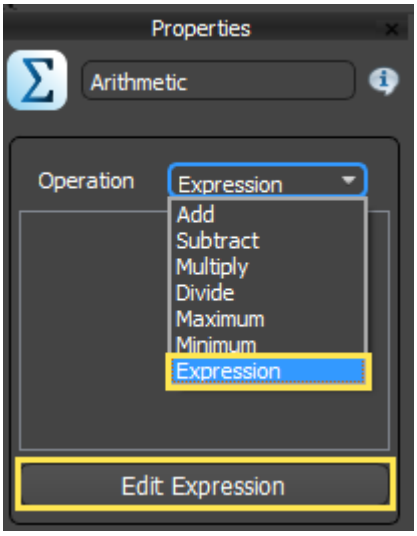
7. The padding value for both of these inputs needs to be half the width of the small tile. Because the small tile is already half the width of the large tile, this means the padding value is a quarter of the width of the large tile. Additionally, to cause the tiles to overlap, it should be a negative value. We could do this by wiring together two arithmetic nodes like this:



If we were to express this in standard mathematical notation then this set of 4 nodes is creating the equation **-1 * (X Size / 4)**.

However we had to use quite a few nodes to create that simple equation, so another option is to use the Arithmetic Operator's **Expression Editor** to enter the equation using standard mathematical notation. To do this add a new **Arithmetic** node to the graph and wire its output to the two **Padding** values you just exported.

8. Wire the *Large_tile* segment's **X Size** attribute to the input of the new Arithmetic node.
9. Select the **Arithmetic** node and change the mode to **Expression** from the Properties Panel

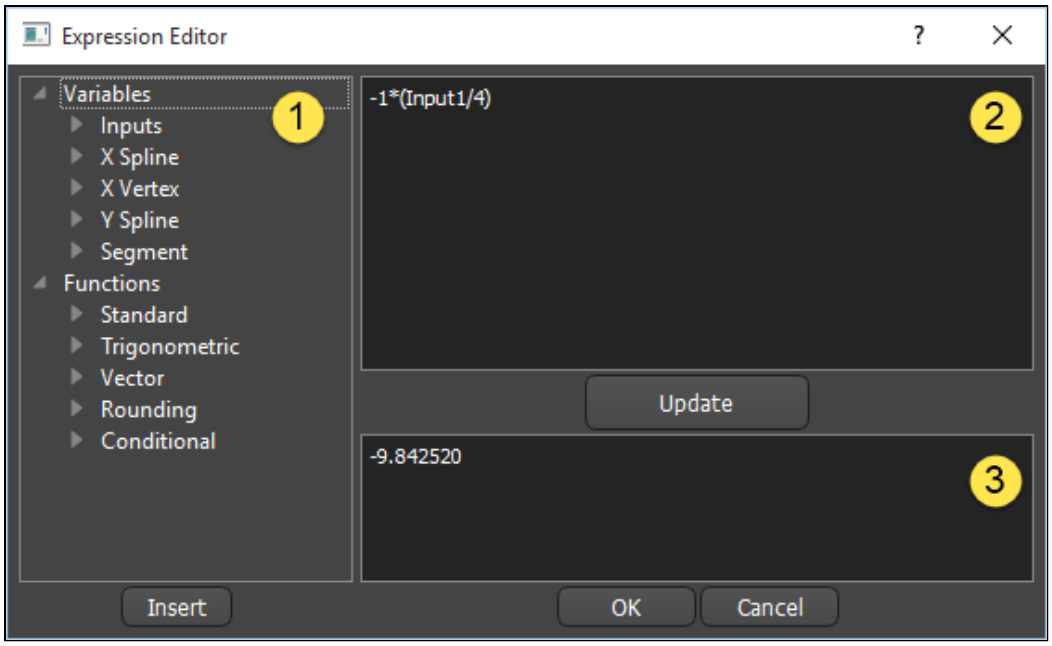


10. Click on Edit Expression to open the **Expressions Editor** window.

Expressions

The expressions editor window.

The expressions editor window is spline into 3 parts:



a. **Expressions pane.** Found on the left, lists all usable variables and functions. Double click a list item to add it to the edit window or single click to display help.

b. **Edit Window.** Used to create and edit the expression, to see the effects of an expression press Update. To construct a variable you can either type it in manually or add variables and functions by double clicking them from the expressions list.

c. **Expressions Help.** This window provides information about the currently selected variable or function or displays the output of the expression.

The editor uses the same syntax as you will find in the expression controllers found elsewhere in Max. This consists of a a straightforward mathematical syntax comprising operators (+, -, *, /, etc.), constants (pi, e, etc.), RailClone's built-in variables, numerical inputs, and functions (including standard, trigonometric, vector, rounding, and conditional functions that take one or more arguments and return a result). To find out which built in functions, operators and constants are available, a full list is included in the **Expressions Pane** on the left hand side of the window. To see an explanation for each function, single click on it and a description and example syntax will display in the **Expressions Help** pane at the bottom of the interface.

You can read more about this and see a full list of supported arithmetic operations in Autodesk's documentation here (with the exception of the predefined variables with variable values, which are not supported). In addition to the functions supplied by Max's expression class, we have added a number of RailClone specific variables. You see in the list on the right of the Expression editor and clicking on each of these will give you a brief description, but for convenience a summary is below.

Inputs

Input1,2,3, ... n These refer to any value wired into the Arithmetic nodes inputs. The number relates to the input slot counting from the top down.

Spline Variables

XSplineCoords,YSplineCoords Refers to the position of the current segment over the Spline on the local coordinates measured from the RailClone objects pivot point. To access the value for a particular axis, the X axis for example, you must append a .x,.y,or .z to the end of the variable, e.g. XSplineCoords.x
XSectionLength, YSectionLength returns the length in scene units of the current section where a section is the path between two Start, End, Corner or Evenly segments.
XSplineLength, YSplineLength returns the length in scene units of the current spline.
XSplinePosition, YSplinePosition returns the position of the current segment on the spline. This is measured not as an absolute measurement in scene units but a decimal value between 0 and 1 that represents the position of the current segment along the spline.
XSectionPosition, YSectionPosition As above but instead of returning the position along the spline, returns the position along the spline section where a section is the path between two Start, End, Corner or Evenly segments.
XSplineType returns 0 if the spline is a line and 1 if it is a curve. Useful for conditional statements.
XSplineMatID returns the material ID applied to the current spline section.

X Vertex

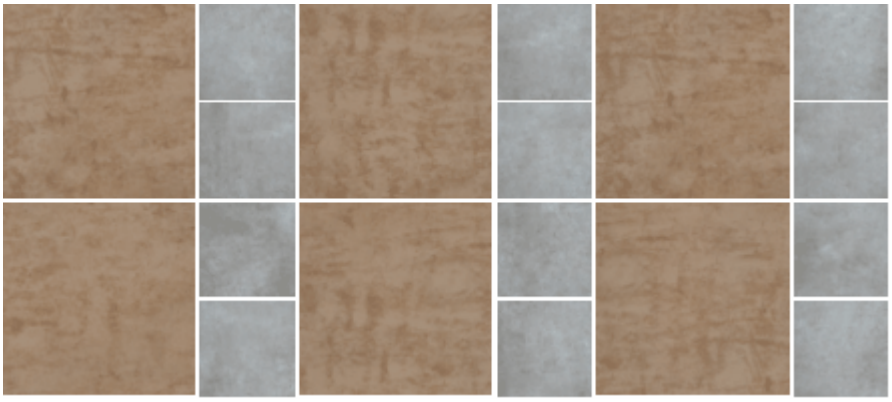
XVertexType returns 0 if the current or previous vertex is of the type smooth, 1 if it is a corner, 2 if a Bezier, and 3 if it is a Bezier-Corner. Useful for conditional statements.

XVertexAngle returns the Angle between X Spline segments on the current vertex measured between 0 and PI radians. (0 - 180 degrees when converted)
XVertexWideAngle returns the angle between X Spline segments on current vertex measured between 0 and 2*PI radians. (0 - 360 degress when converted)

Segments

SegmentXCounter, SegmentYCounter returns the current segment's index along the path. Note that there are two separate counters: one for default segments, and another for Start/End/Corner/Evenly combined.

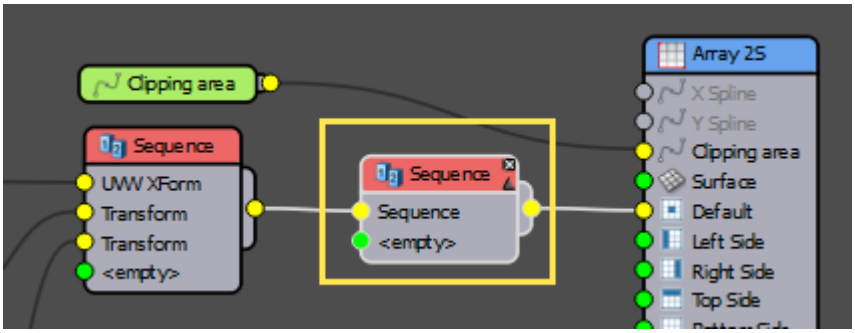
11. Enter the expression ***-1*(Input1/4)***. In this expressions Input1 is the value piped in by the node connected to the first input.
12. The style will now be updated with an alternating pattern of one large tile, followed by two small tiles one above the other. If you edit the size of the large tile, the small tiles also adjust automatically.



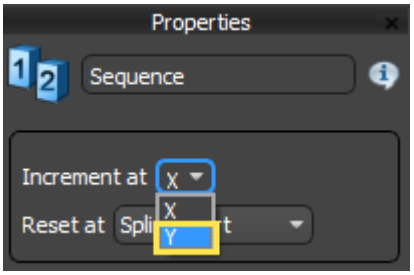
Exercise: Creating the horizontal rows of small tiles

The style is nearly complete, we just need to create the alternating horizontal rows of small tiles. To do this, follow these steps.

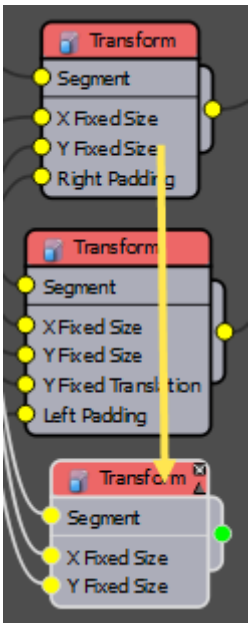
1. Create a new **Sequence** operator and wire it between the existing **Sequence** operator and the **Generator's Default Input**



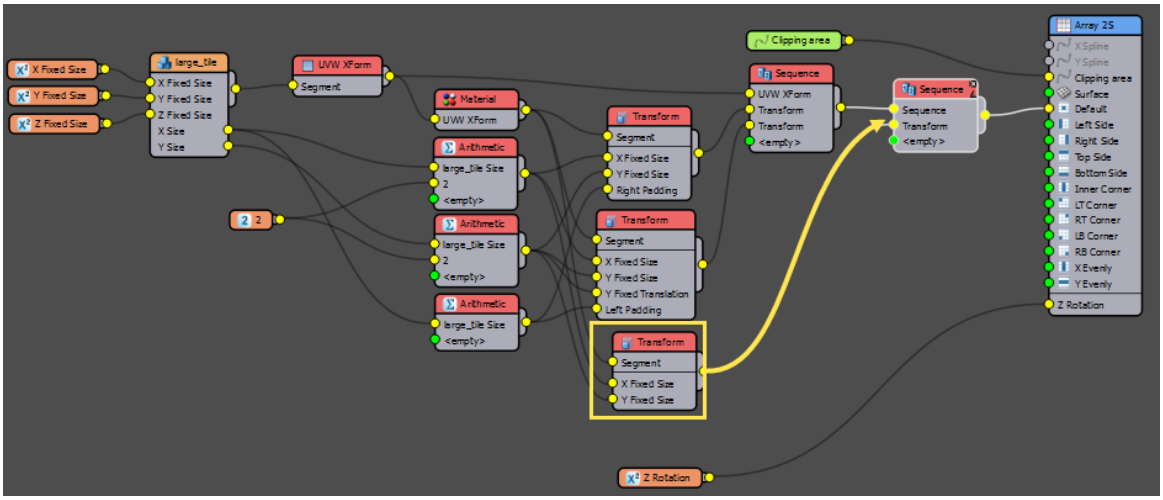
2. Select the new Sequence operator and change the mode to **Increment** on **Y** from the **Properties** panel.



3. We don't need the segments in these rows to overlap, so Copy and past the first of the existing Transform nodes and then right click and turn **Off** the exported Padding parameter.



4. Wire the third (new) **Transform** operator to the new **Sequence** operator's **second** input.



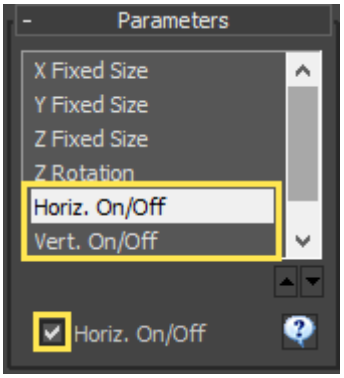
5. You now have a style with alternating rows and columns of small and large tiles with adjustable size on the X, Y, and Z axis and Rotation!

Exercise: Turning on/off features

Finally we'll add one last function to the style and demonstrate how to disable nodes so that features of a style can be turned on and off from the modify panel. In this example we'll provide two tick boxes that turn on or off the horizontal or vertical rows of small tiles by enabling and disabling the two sequence operators.

> ⓘ When a node with multiple inputs is disabled, the geometry wired to the first input is passed on to the next active node in the graph. Any geometry in the other inputs is ignored. In this example we are using Sequence operators so disabling them will allow the first input to continue to the generator, disabling the pattern caused by the node.

1. Right click on both Sequence generators and **Export** the **On/Off** parameters.
2. For each of these exported parameters create a new **Numeric** node and set the **Type** to **Boolean**.
3. Give each one an appropriate name and wire it to the exported On/Off inputs. You can now turn these nodes off by clicking the check-boxes found in the **Parameters** rollout of the **Modify** panel.



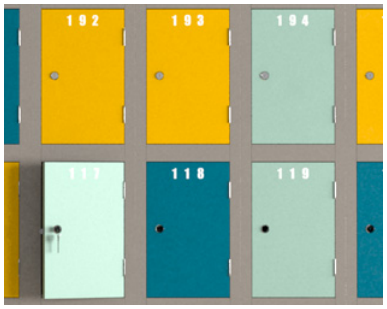
This completes the tiling style. If you've followed all the exercises in this chapter you will have exported parameters and attributes, created numeric nodes to control aspects of the style from the modify panel and used constant nodes to set fixed values. You will also have used the expression editor to create a simple equation and minimise the number of nodes that need to be created for common mathematical functions. In the next chapter we'll look at material IDs and UVws and use exported values to take more advanced control material ID randomisation.

3.2 Related Tutorials

creating_parametric_shelving

Creating Bookshelves

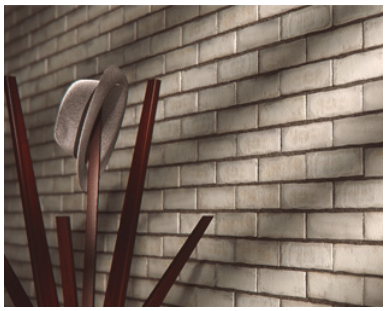
In this Tips & Tricks tutorial we look at how RailClone can be used to create procedural furniture. Follow this exercise to learn how to export many properties to allow a user to adjust the height, width and shelf distance of a bookshelf directly from the modify panel



creating_number_sequences

Creating Number Sequences

For some styles it is necessary to create numbers that increment for each segment. In this Tips and Tricks episode we look at two ways to create sequences of numbers. The first example illustrates how to create a series of numbers by cycling through geometry. In the second part we manipulate a segment's material IDs using expressions to create numbers using multi-sub object materials.



mapping_tiles_and_bricks

Mapping Tiles and Bricks

In this advanced exercise we look at a time saving graph-snippet for mapping segments and randomizing them with individual tiles or bricks from textures. This tutorial features numeric, constant, arithmetic nodes and exported attributes.

building_bridges

Building Bridges

In the second part of this intermediate tutorial on creating bridges we demonstrate how to use the arithmetic node and expressions to create a bridge with a deck that is able to deform on the Z axis with piers that automatically scale to support it.

Documentation

For more information please see the Segments, 2D arrays - Generator A2S, 1D arrays - Generator L1S, Exporting Parameters, and Arithmetic sections of the online documentation.

4 4 - How to Control Material IDs

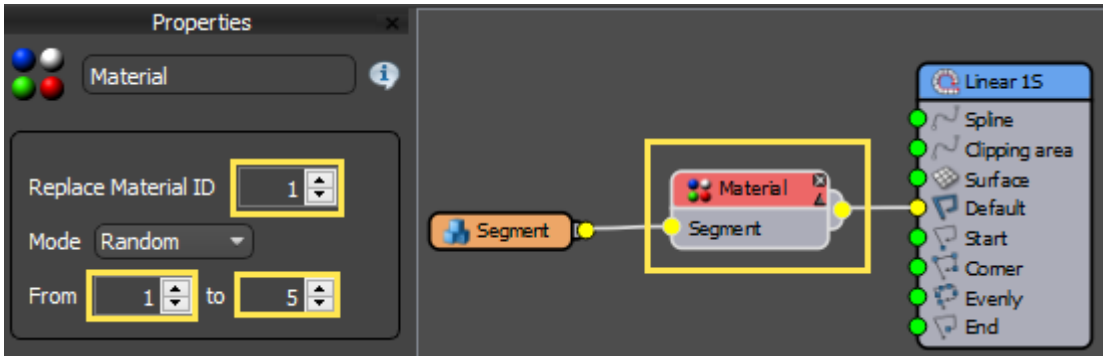
RailClone provides a powerful set of tools to control the UVW mapping and Material IDs of segments. Not only is RailClone able to Randomise material IDs and adjust the position, rotation and scale of existing UVW coordintates, but it is also possible to generate new box mapped UVWs automatically that follow the source path. This is ideal for situations where you need the texture to bridge multiple adjacent segments and create the illusion of a seamless unified surface.

Compatibility

V-Ray is required to maintain instancing when using UVW Xform, Automatic Box Mapping and Material operators. Other renderers are supported by disabling instancing. To do this, turn off **Display->Use Instancing Engine**. To see a full list of which features are supported by your renderer, check the compatibility list.

4.1 Randomising and Sequencing Mat IDs

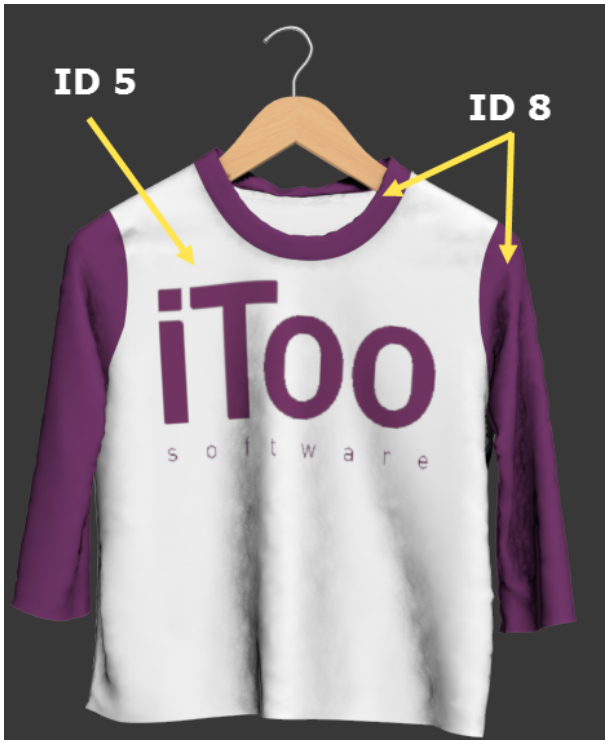
RailClone includes a Material operator that allows you to select a material ID and randomise or sequence it between a minimum and maximum value. To use it, all you need to do is wire a new Material node between a segment and the generator, choose the material ID to randomise using the Replace Material ID value, and set the Range using From and To values:

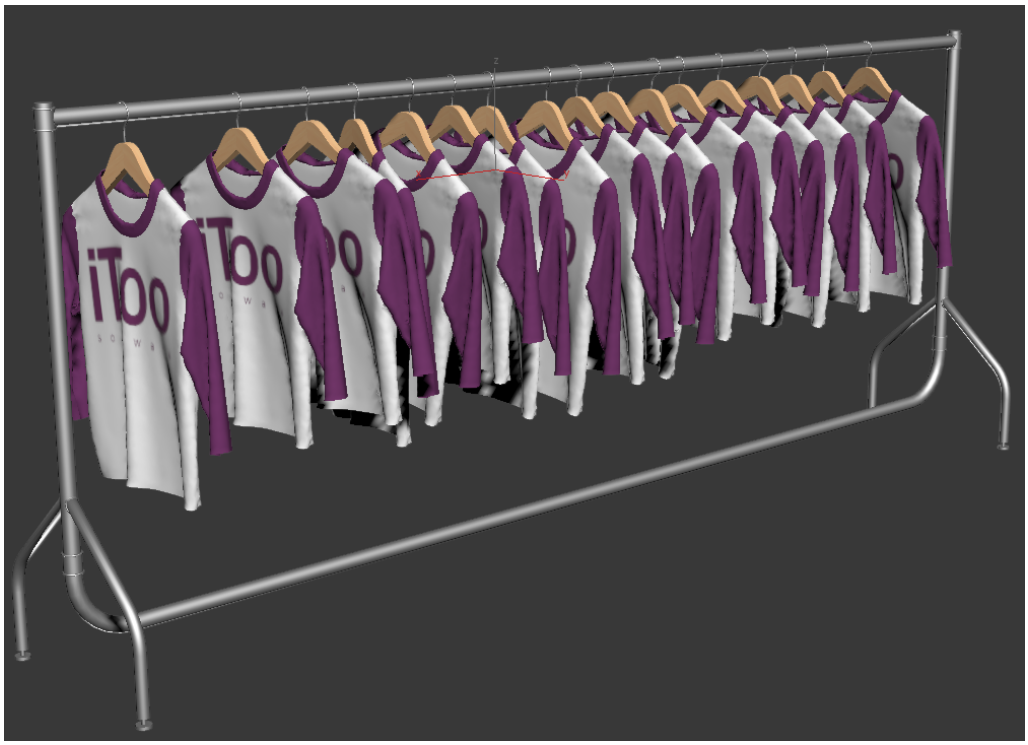


Exercise: Randomising Shirt Colours

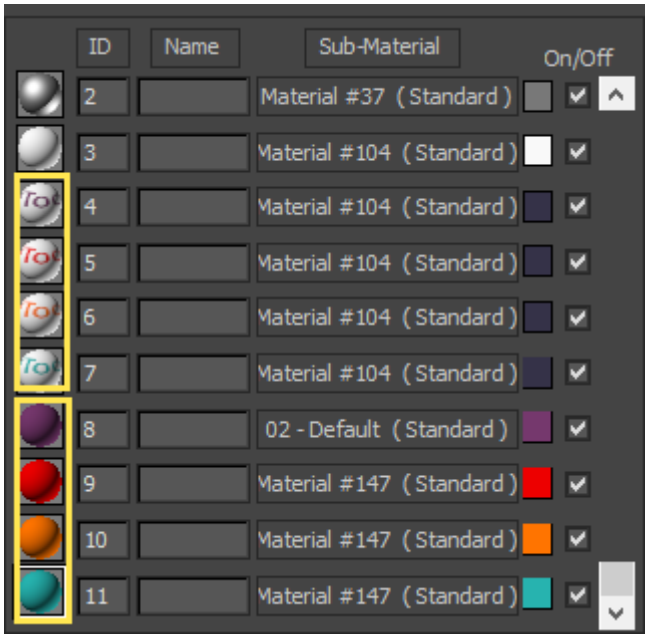
To see how this works in practice we will edit a style to randomise the colour of t-shirts between 4 colours. To see the existing style, open *chapter_4_randomise_ids_start.max* from the downloads for this tutorial. This style uses two segments: the t-shirt on the hanger, and a separate hook.

When randomising IDs it's important that the source geometry and materials are set up correctly. You'll need to know the material IDs of the elements you wish to randomise and also the range of IDs in the multi/sub object material that you intend randomise between. In the following example the T-Shirt has separate ID's for the logo colour and the sleeves.





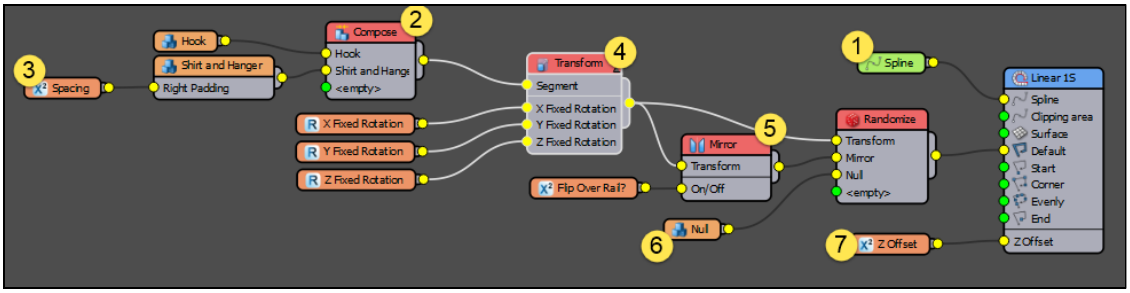
The material is created so that each colour is divided into two blocks of 4 colours, with the logo between ID 4 and ID 7 and the sleeves between ID 8 and ID 11



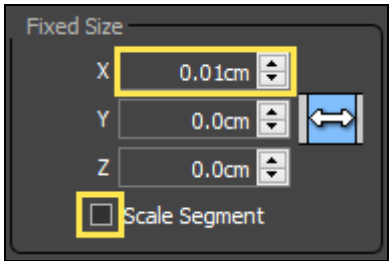
With that information, we can update the style to randomise the colour of the sleeves.

1. There are two shirt styles already in the scene, select *rc_shirts_1* (we'll use the other one for the next exercise).
2. Open the **Style Editor**.

The style has already been set up so that only the Material operators need to be added. However if you want to know a little more about how this style works then the notes below should help.

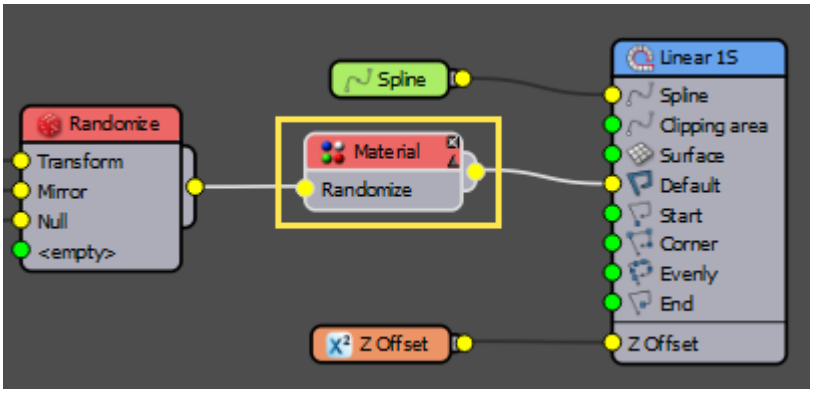


1. The t-shirt style uses a simple L1S array driven by a spline path.
2. Thought the shirt and hanger are a single segment, the hook has been separated so that its rotation can be randomised for a little extra variation. To stop the rotation affecting the position of the adjacent segments, a small Fixed Size value is used with Scale Segments turned off.

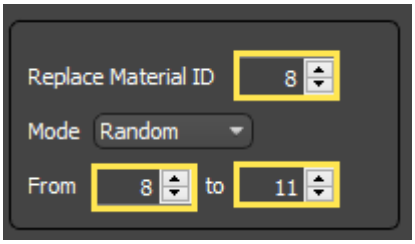


3. Right padding is used on the shirt segment to create a gap between hangers. This has been exported so you can adjust it from the Modify panel.
4. The Transform operator is used to subtly randomise the rotation. Fixed Rotation is used with Random number operators to generate the same values for the hook and the shirt. Using the built in Randomise Transform settings would result in different values for these segments and they would no longer align correctly.
5. The mirror operator is set to X and Y to flip the hanger over the railing. This is to add greater variety and "messiness" but the on/off parameter for the generator has been exported so you can disable it if you choose.
6. This is an empty segment with a fixed size setting of 15cm. It is wired to a randomise operator along with the geometry to randomly generate gaps in the array.
7. The generator's Z Offset has been exported so that you can adjust the hanging height of the shirts to match the rail.

1. Create a new **Material** operator and wire it between the existing **Randomise** operator and the **Generator**.



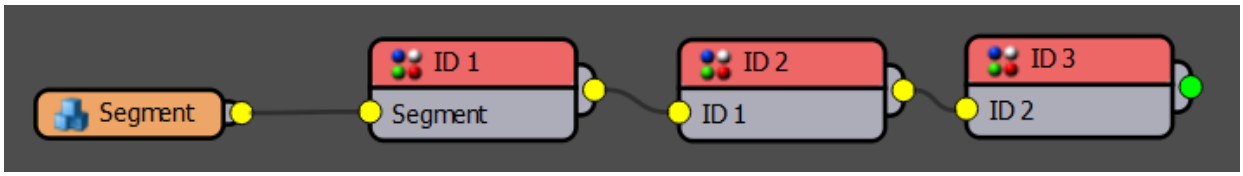
2. Select the **Material** node and go to the properties panel. The sleeves use Material ID **8** so enter this value for **Replace Material ID**.
3. To set the range enter **8** as the **From** value and **11** as the **To** value.



The sleeves are now randomised between 4 colours!

Randomising multiple IDs

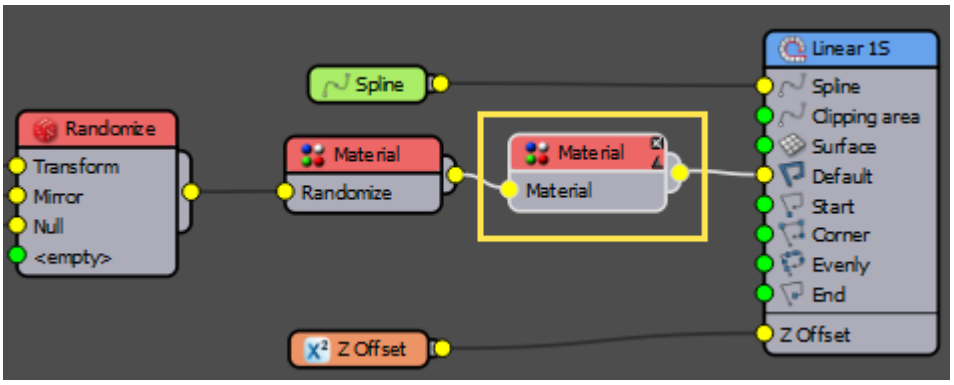
As you can see, the Material operator can only randomise or sequence one ID at a time. If you would like to randomise multiple IDs then this can be easily achieved by wiring together a sequence of material operators as shown below.



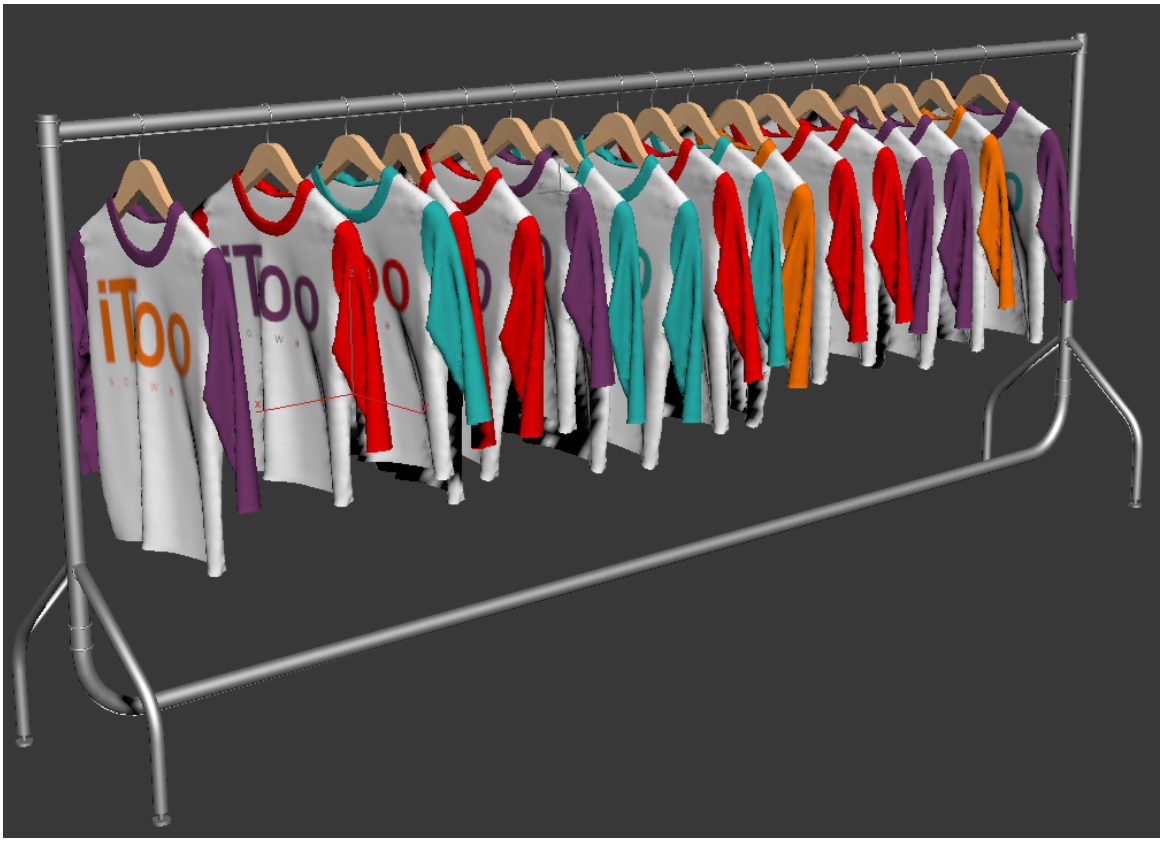
Exercise: Randomising the logo colour

In the T-Shirt example we have randomised the sleeves but not the logo. To randomise this as well, follow these steps.

1. Wire a second **Material** operator between the existing Material node and the generator.



2. Change the new Material operator's **Replace Material ID** value to **4**. Set the **From** value to **4** and the **To** value to **7**. The logo and the sleeves are now randomised!



Randomising by RailClone object, spline, spline section or segment

In the clothes example you will notice that the sleeves and the logos are randomised separately so the colours often do not match. If you need to synchronise the colour of different material IDs on the same segment then you will have to set up the two Material operators slightly differently. Instead of using the From and To values to set the randomised range, you can export the values and wire them to a Random number node.

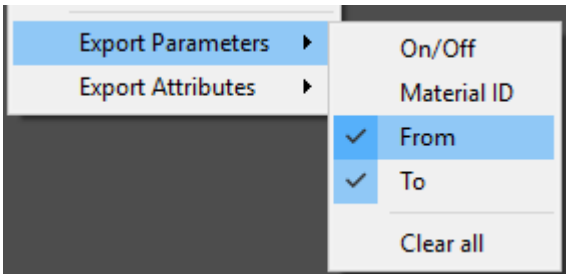
The Random Number node is useful for those parameters that do not already have a randomisation feature, or if you want more control over when a new random number is generated. For example Segment and Transform nodes already have parameters to randomise the translation, rotation and scale of a segment, but they generate a new number for every segment. What if you wanted to generate a new value only at the start or for each spline, section, or y evenly row? For this you can use the Randomise node to precisely control when a new random number is generated, and because the value is exposed it can be wired to multiple nodes so that values can be synchronised or offset using arithmetic operations.

Exercise: Matching the sleeves and logo colour

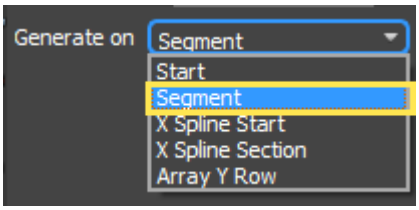
In the shirt example we have 2 block of material ids: The logo IDs are between 4 and 7, and by adding 4 to these values we get the corresponding sleeve colours, between 8 and 11. To make sure they are always in sync we need only create a single Random number node that picks a number between 3 and 7 for the logo and then wire this to an arithmetic node that adds 4 to get the corresponding sleeve colour:

To test this, select the second RailClone object, open the style editor and then follow these steps:

1. Wire a new **Material** operator between the existing **Randomise** node and the **Generator**
2. Set the **Replace Material ID** value to 4
3. Right click on the Material node and export **From** and **To**.



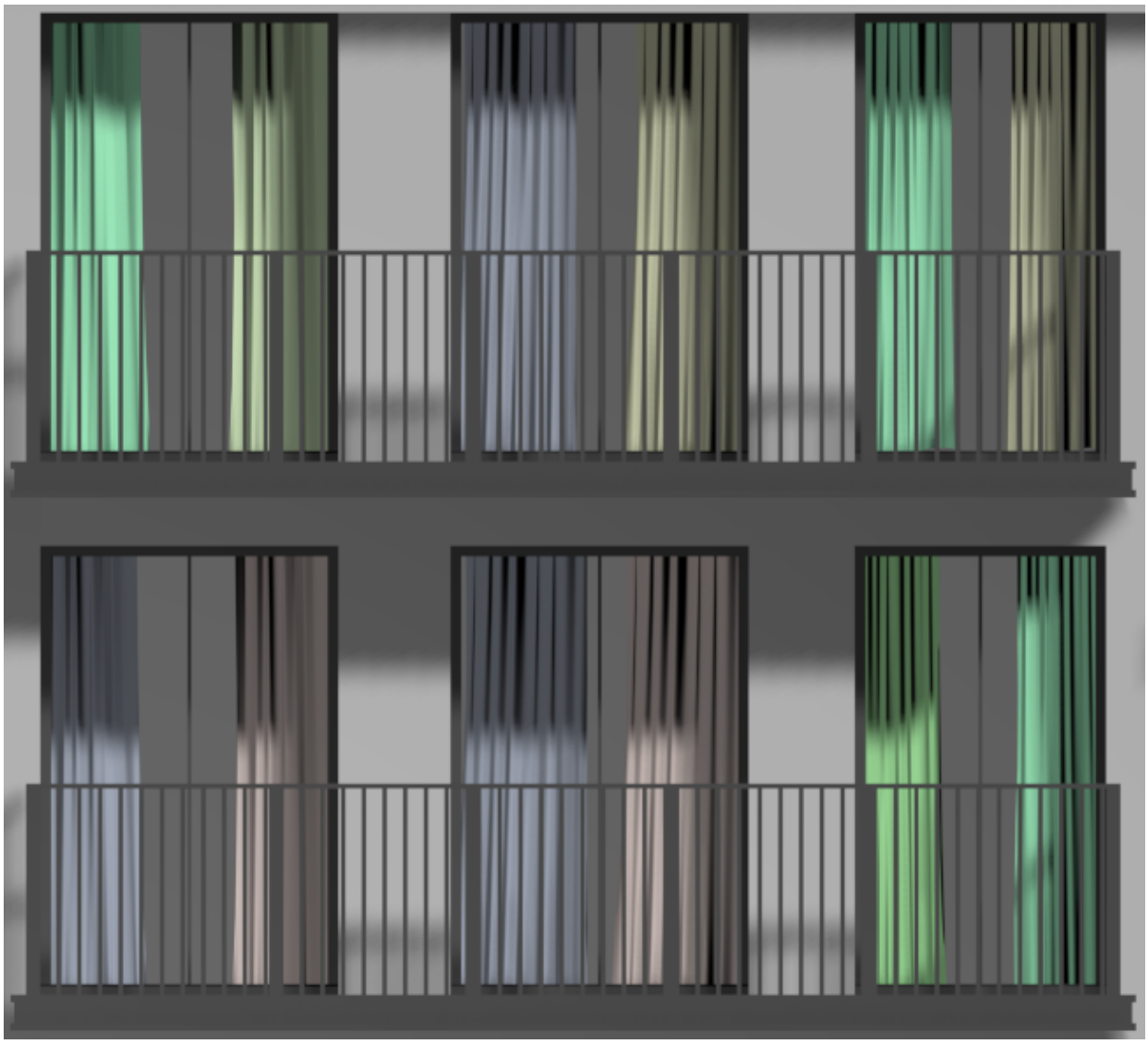
4. Create a new **Random** number node and connect it to the **Material** node's **From** and **To** inputs.
5. From the Properties panel, set Random number node's **Min** value to **4** and **Max** value to **7**.
6. Set the Random Node's **Generator On** mode to **Segment**. The logos should now be randomised.



7. To randomise the sleeves, wire a second new **Material** operator between the existing Material node and the generator, then right click on the new Material node and export **From** and **To**.
8. Create an **Arithmetic** node and wire the output to the new Material node's **From** and **To** inputs. Leave the arithmetic node set to its default **Addition** mode.
9. Wire the **Random Number** node to the **Arithmetic** node's first input.
10. Create a new **Constant** node and set its value to **4**. Wire this to the **Arithmetic** node's **second** input.
11. You will now have a style with randomised colours, but this time the logo and sleeve colours match!

Exercise: Randomising by spline and spline section

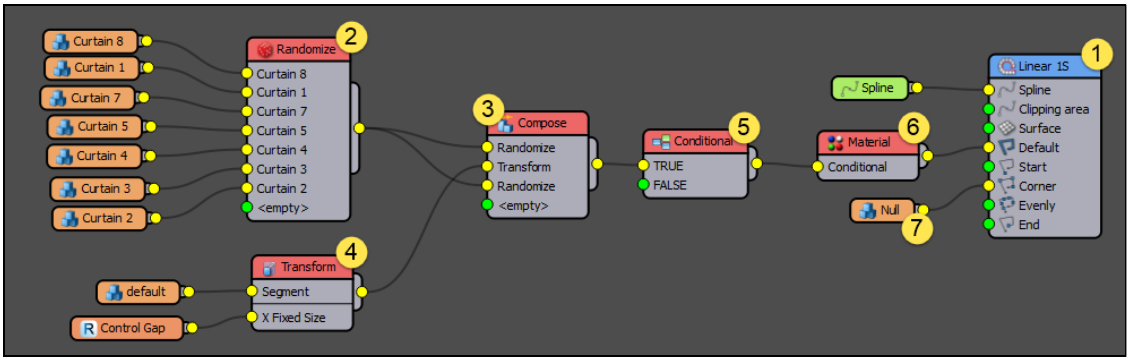
In a second example, the Randomise operator is used to add different colours to curtains to add some variety to an exterior view. As you can see the same problem has arisen where each individual curtain takes a different random colour.



It is unlikely that somebody would use different coloured curtains in the same window, so let's fix this using the technique demonstrated in the t-shirts exercise above. We'll then look at how we can ensure the curtains are the same in every room as well as every window.

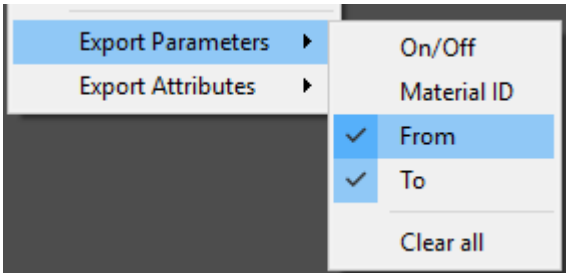
1. Open *chapter_4_curtains_start.max* from the downloads for this tutorial.
2. Select the *Curtains* RailClone object and open the Style Editor.

Once again, the majority of this style has already been set up so that only the material operators need to be added. However if you want to know a little more about how this style works then the notes below should help.



1. A linear array is used driven by a spline object which uses a separate spline sub-object for each room. The windows are identified by a spline segment with an ID of 1. Sections of a room that are not a window are indicated by a spline segment with an ID of 2. This is so that the same splines can be re-used to drive the balconies.
2. Seven curtain segments are wired to a Randomise operator to create variation.
3. The curtains are then wired to a Compose operator twice, with an empty "null" segment between them to create a gap. This set of 3 segments is stretched on each splines section by setting the Generator's default mode to Count and setting the Count value to 1.
4. The size of the gap between the curtains is randomised by wiring a Null segment to a Transform operator and exporting the X Fixed Size parameter. This value is then driven by a Random Number node.
5. The Conditional operator limits the segments so that they only appear on spline segments with an ID of 1.
6. The Material operator randomises the materials, **this is the part we need to change in order for this style to work correctly**.
7. There is a null segment in the corner input to force a break in the spline where the ID changes. Without this a single set of curtains would be stretched along the full length of the spline.

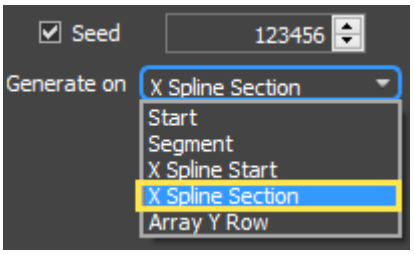
1. Right click on the **Material** node and export **From** and **To**.



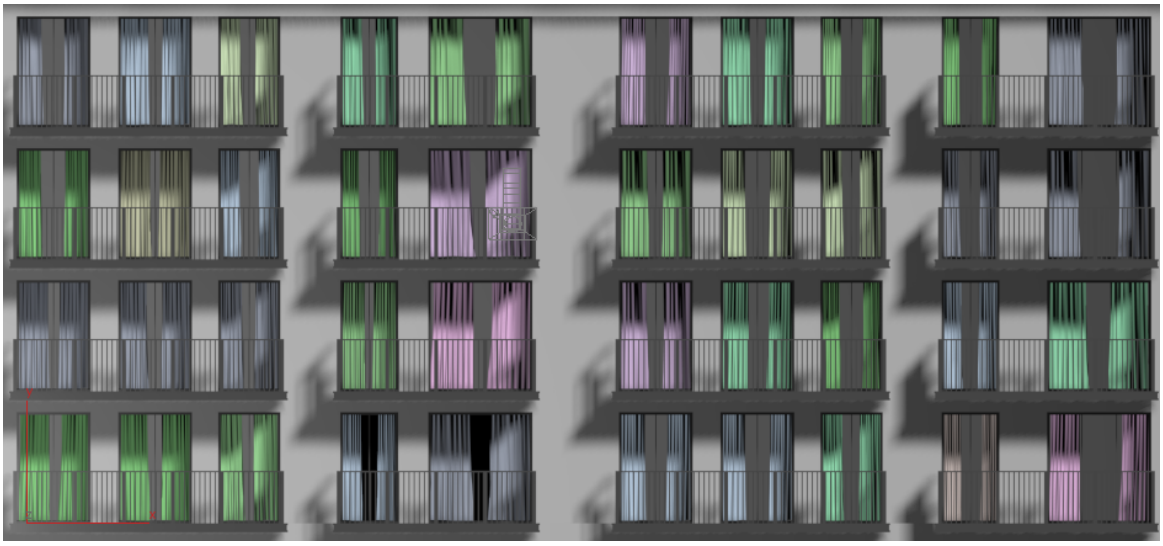
2. Create a new **Random** number node and connect it to the **Material** node's **From** and **To** inputs.

3. This time there are 10 materials in the multi-sub object material for the curtains so go to the **Properties** panel, set the **Random** number node's **Min** value to **1** and **Max** value to **10**.

4. Set the Random Node's **Generator On** mode to **X Spline Section**.

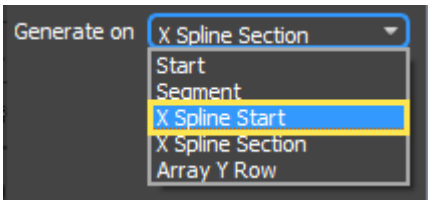


5. Because the curtain in each window are on the same spline section they will now match..

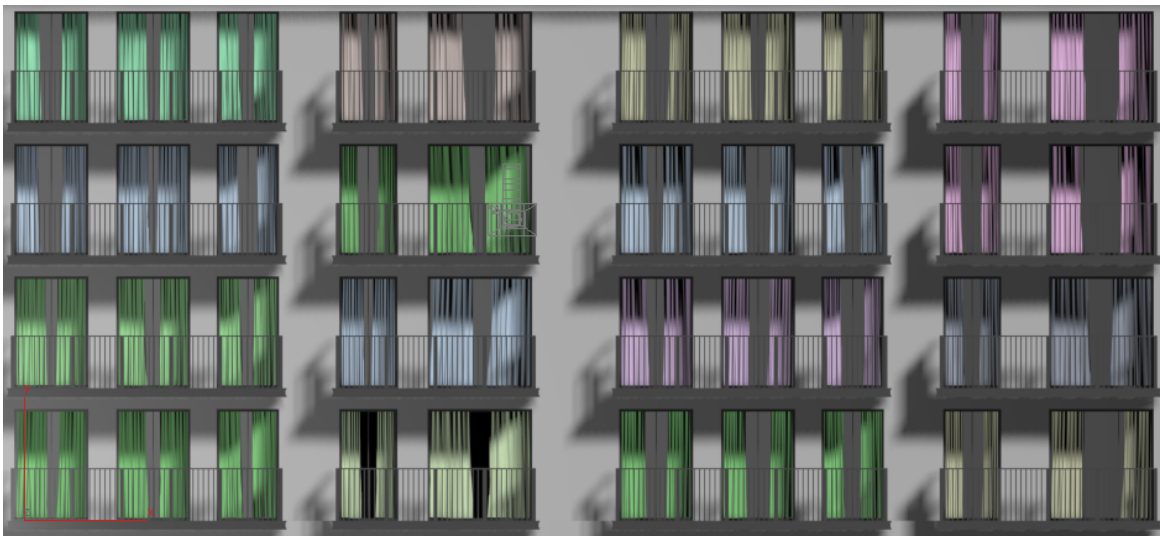


This looks better but it is likely that several of those windows are shared by a single room.

For the sake of this exercise, let's assume that each balcony marks a single room. This file has been set up so that each balcony and set of windows and curtains is using its own spline segment. Therefore to ensure all the curtains within the same room are the same we can simply change the Randomise operator's Generate On value to X Spline Start.

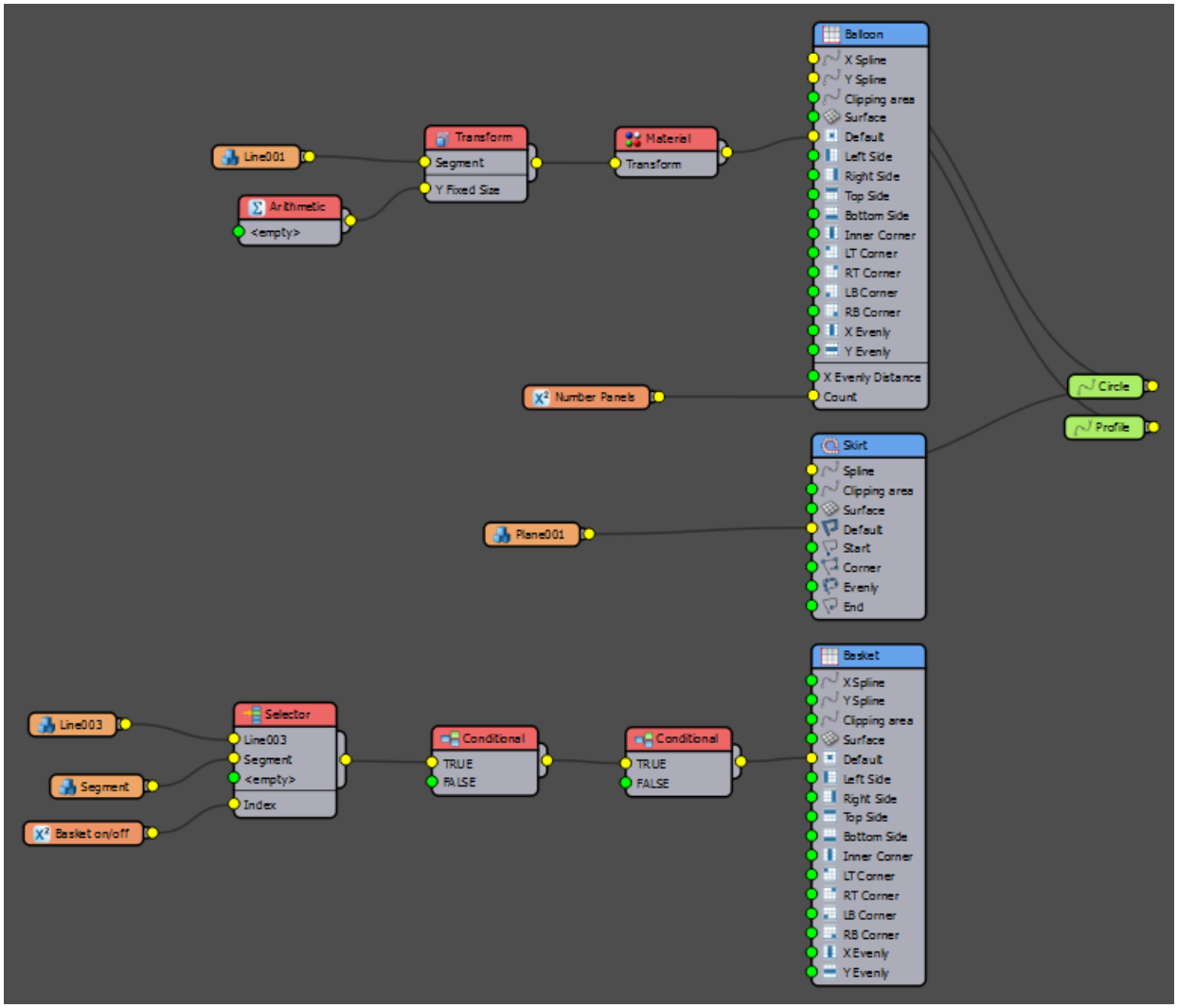


6. Each room now shares curtains of the same colour. You can use the Randomise operator in this way for any Parameter in RailClone. In this example we have used it to control when to randomise Material IDs, but you could also use it for transforms, Selector > index, and anything else with a exportable parameter.

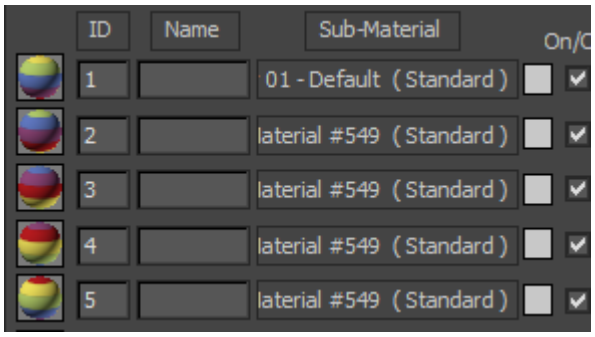


Exercise: Using the Randomise operator to create a sequence of material IDs.

A less well know feature of the Material operator is the ability to cycle through a sequence of material IDs. In this exercise we'll create a pattern on a hot air balloon using this feature. The RailClone hot air balloon uses 3 generators: An A2S generator for the balloon itself with the shape generated using two spline, an L1S generator to create a skirt around the bottom, and finally another A2S to position the basket. Click on the image below to see a larger view of the node tree.



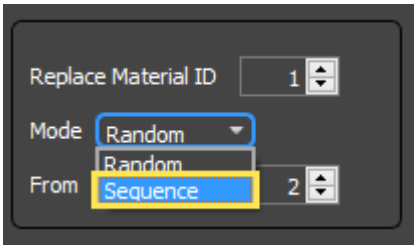
The balloon has a material applied with 5 striped patters in material ID slots 1 - 5. The texture is the same for each material but has been vertically offset by an fifth each time it is used. By creating a sequence of these 5 materials we'll add a pattern to the balloon.



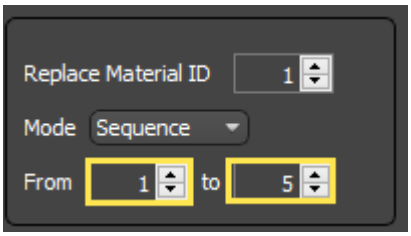
Here's how the balloon looks before we apply the material operator.

To create the pattern:

1. Open *chaper_4_balloons_start.max* from the downloads for this guide.
2. Select *rc_balloon_1* and open the Style Editor.
3. The topmost A2S generator is responsible for creating the Balloon. Create a new **Material** operator and wire it between the **Transform** operator and the **Generator's Default input**.
4. Select the **Material** node and change the **Mode** to **Sequence**.



5. Enter **1** for the **From** value and **5** for the **To** value.



By sequencing the IDs in this way you should now have a pattern of 5 offset textures on the balloon.

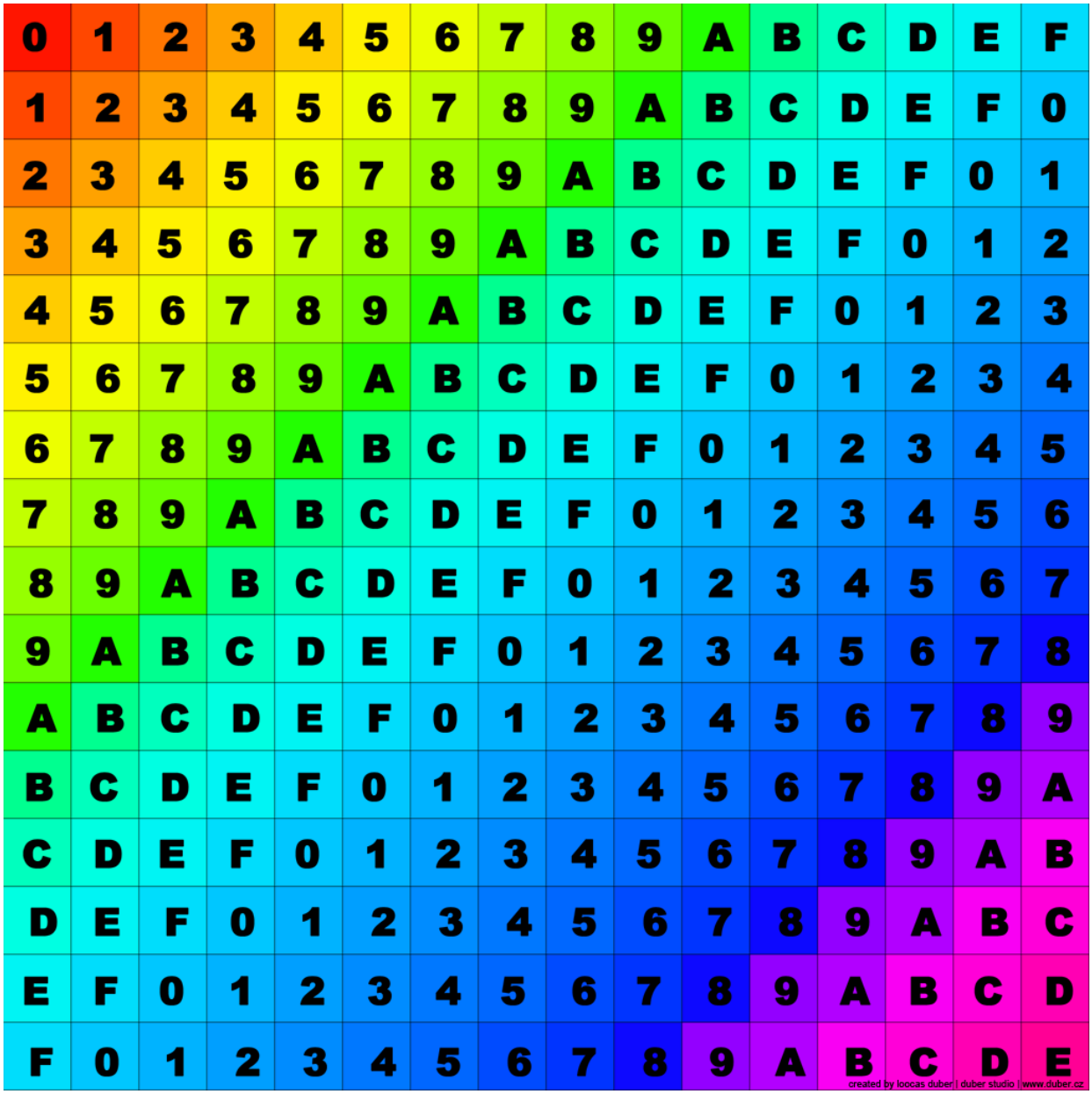
We'll return to the balloons in the next section which looks at UVW XForm, another powerful modifier that enable you to adjust existing UVW coordinates on segments.

4.2 UVW XForm Operator

The UVW XForm operator allows you to easily edit and randomise the offset, tiling and rotation of existing UVW coordinates. It's particularly useful when you need a lot of texture variation, but you don't want to have to create time-consuming Multi/Sub-Object materials and randomise material IDs. (Of course for even more variation - both techniques can be used together!)

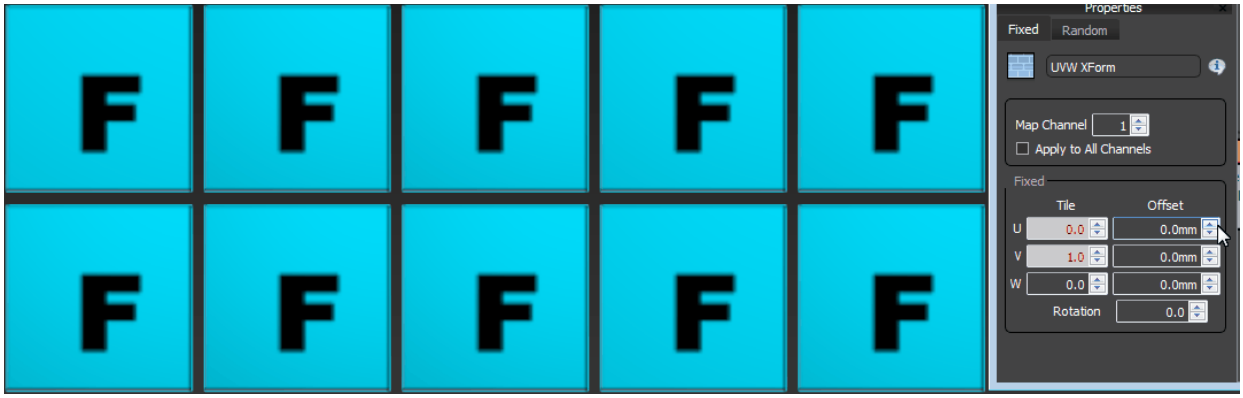
Editing existing UVWs

To help illustrate how the operator works, in the following gifs the texture shown above has been applied to a simple tile mesh. A single tile has been mapped before adding it to the RailClone style so that only one of the 256 coloured alphanumeric tiles are displayed.

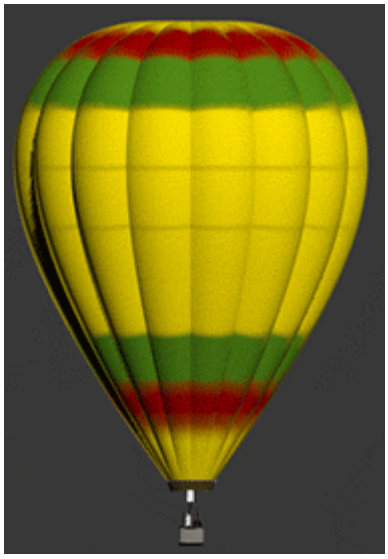


You can choose to edit any **Map Channel,** or if you'd prefer to affect all the channels, turn on **Apply to All Channels.** If you want to adjust a limited selection of channels, for example map channels 1 and 3, you can use multiple UVW XForm operators wired together in series.

The **Fixed** group allows you to edit the **Tiling**, **Offset**, and **Rotation** of the existing mapping. There are independent controls for each of the U,V and W axes.



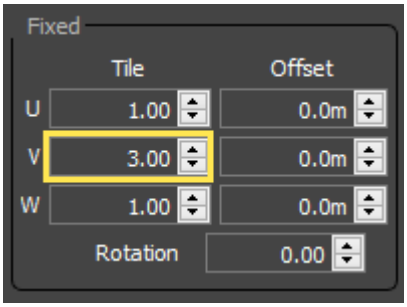
In the following short exercises we'll return to our balloon models and use the UVW XForm operator to edit the patterns. In all these examples we use only a single texture instead of the 5 required to create the pattern in the previous exercise.



Exercise: Editing UVW Tiling

1. Open *chaper_4_balloons_start.max* from the downloads for this guide.
2. Select *rc_balloon_2* and open the Style Editor.
3. Wire a new **UVW XForm** operator between the the **Transform** operator and the **Generator's Default input**.

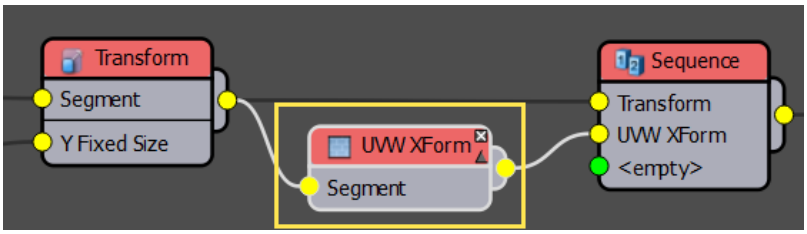
4. Select the UVW XForm node and set the **Fixed** > **Tile** > **V** value to **3**



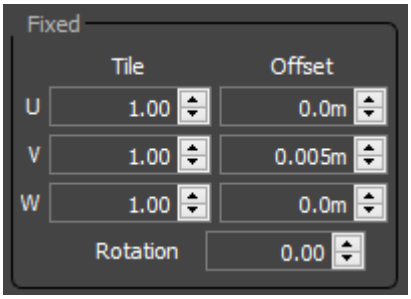
Exercise: Editing UVW Offset

In this example we use the sequence operator in conjunction with UVW Xform to create the pattern

1. Select *rc_balloon_3* and open the Style Editor.
2. Wire a new **Sequence** operator between the the **Transform** operator and the Generator's **Default** input.
3. Create a new **UVW XForm** operator and wire it to the **Sequence** operator's second input. Wire the **Transform** operator to the UVW Xform node's input.



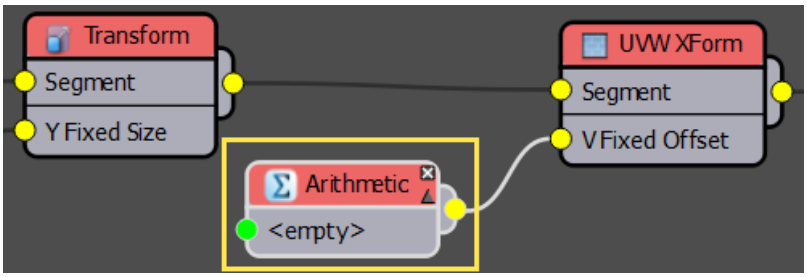
4. Select the **UVW XForm** node and set the **Fixed** > **Offset** > **V** value to **0.005m.**



Exercise: Using Expressions with UVW Offset

The last example was straight forward but what if we needed to offset the pattern many times. Rather than create and edit a lot UVW Xform nodes we can use just one with an expression.

1. Select *rc_balloon_4* and open the Style Editor.
2. Wire a new **UVW XForm** operator between the the **Transform** operator and the **Generator's Default input**.
3. Right click on the **UVW XForm** node and export **Fixed** > **Fixed Offset** > **U**.
4. Create a new **Arithmetic** operator and wire it to the **UVW XForm** node's **U Fixed Offset input**.



5. Select **Arithmetic** node and change the Operation to **Expression**.
6. Click **Edit Expression** to open the editor window.
7. From the list of options on the left, find and double click on **SegmentXCounter**. This will return the number of the current segment.
8. After this enter ***0.25*** so the the finished expression reads

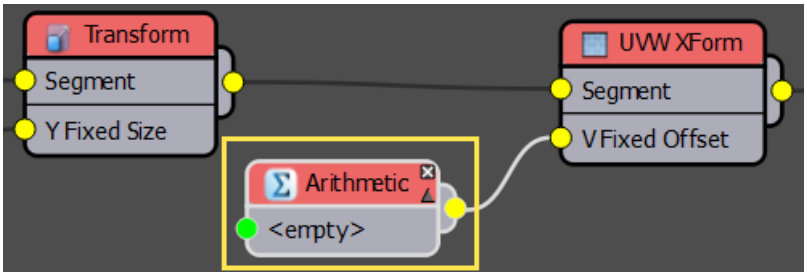
   SegmentXCounter*0.25

   This expression will offset the UVWs by 0.25 for every segment to create the pattern.
9. Click OK to close the Expression editor.

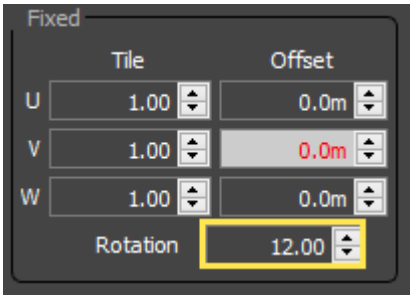
Exercise: Editing UVW rotation

In this exercise we use the technique from the previous example, plus UVW rotation to create the spiralling pattern

1. Select *rc_balloon_5* and open the Style Editor.
2. Wire a new **UVW XForm** operator between the the **Transform** operator and the **Generator's Default input**.
3. Right click on the **UVW XForm** node and export **Fixed** > **Fixed Offset** > **U**.
4. Create a new **Arithmetic** operator and wire it to the UVW XForm node's **U Fixed Offset input**.

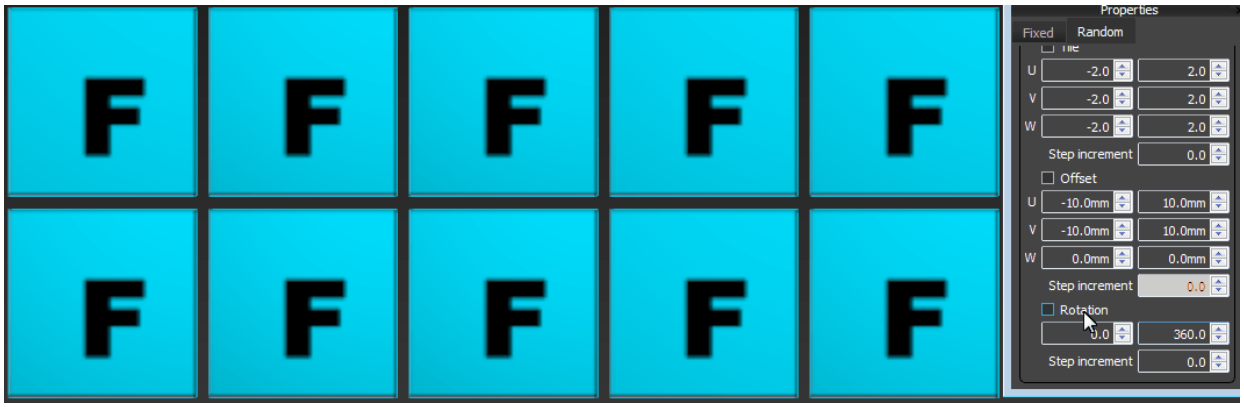


5. Select the **Arithmetic** node and change the Operation to **Expression**.
6. Click **Edit Expression** to open the editor window.
7. Enter the expression ***SegmentXCounter*0.2***.
8. Select the UVW XForm operator and enter **12** in the **Rotation** property.



Randomising existing UVWs

The UVW XForm operator also includes powerful randomising features. Using these you can create a tremendous amount of variation from a single map. There are separate controls to randomise the tiling, offset and rotation. All you need to do is click the checkbox and set a minimum of maximum value.

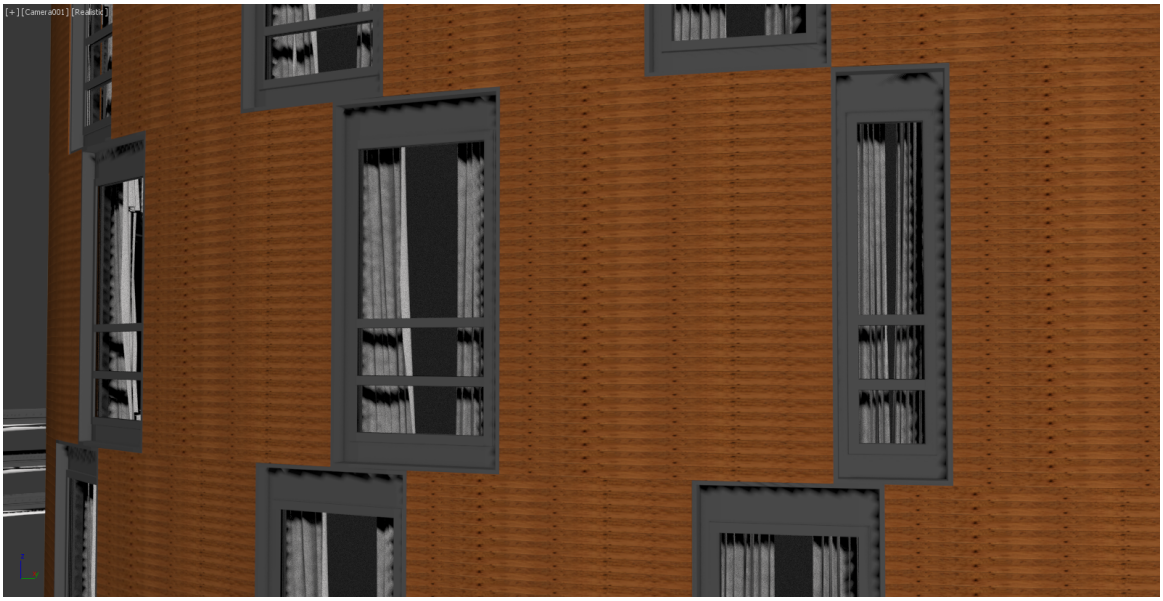


For an example of this technique in action, let's take a look at a cladding style.

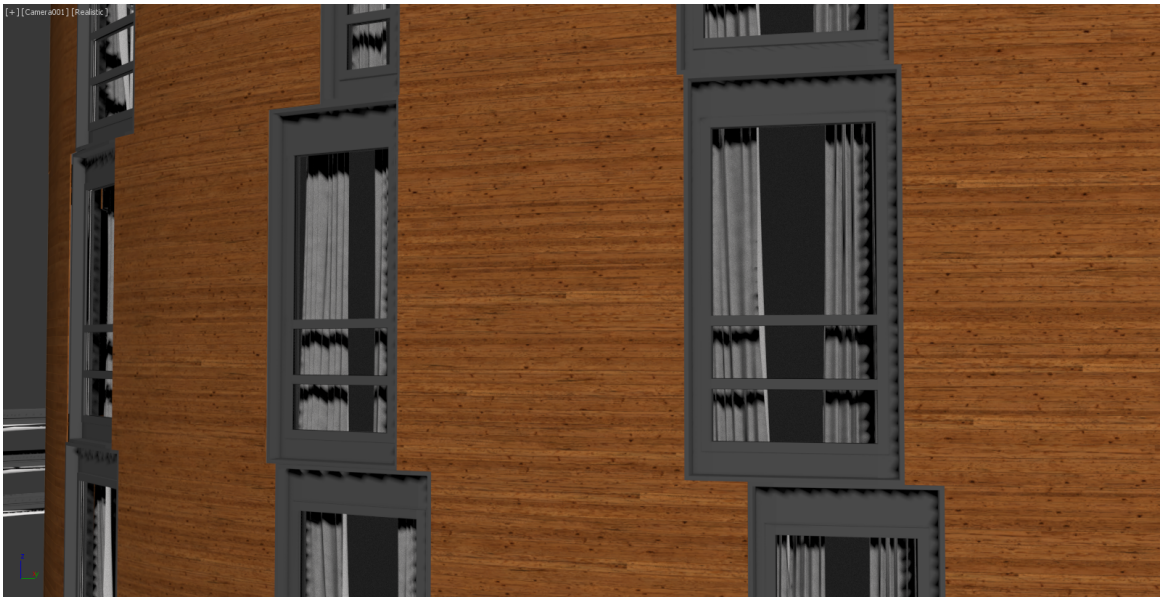
1. Open chapter_4_wood_facade_start.max from the download for the tutorial. The wooden parts of the facade are using the texture show below.

2. The box segment has been mapped prior to importing into RailClone so that only a small portion (a single plank) is displayed.

3. Here's how this looks on a façade, as you can see the texture repetition is obvious. Let's fix it...



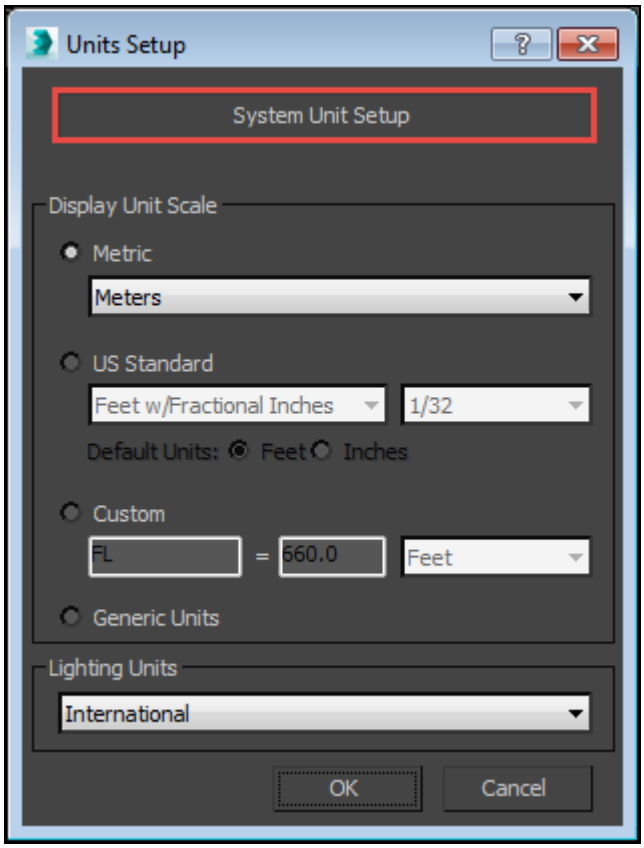
4. This scene actually has two RailClone objects, one nested inside the other. The wooden cladding is it's own RailClone object, called *rc_wood.* This is so that each plank can be randomised individually. *Rc_wood* is added as a segment inside *rc_facade* to combine it with the windows. In this example we want to edit the wooden part of the facade so select *rc_wood* and open the Style Editor.
5. Add a new **UVW XForm** operator between the existing **Transform** node and the generator's **Default input**.
6. Go to the **UVW XForm** node's properties and click the **Random** tab to open the settings
7. Activate the **Offset** checkbox and enter -**0.01m** for the **U** and **V minimum** values and **0.01 m** for the **maximum** values.
8. Each plank's UVWs will now be randomly offset, making it much harder to spot texture repetition.

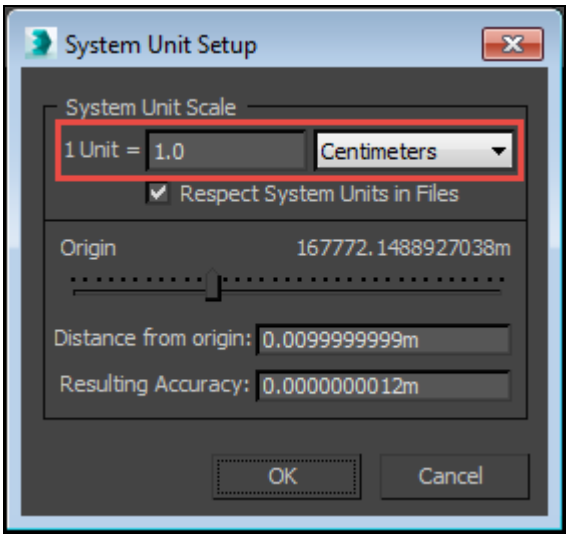


Why 0.01m?

The spinners in the UVW XForm operator use scene units. If you're using real world map sizes this makes sense and behaves as you'd expect, but if you're using standard mapping, it's not necessarily obvious what measurements correspond to 0 - 1 in UVW

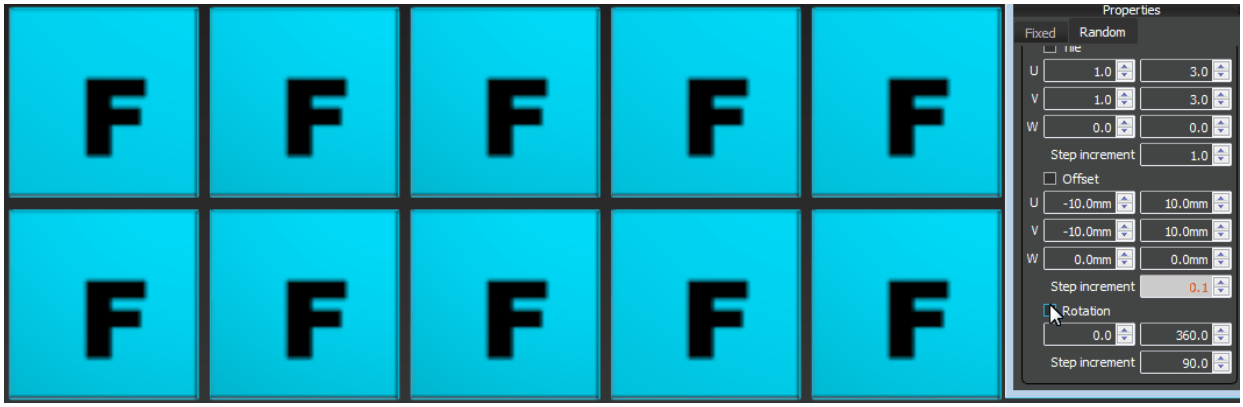
space. The answer is that 0 - 1 UVW space corresponds to 0 -1 system units. To find out what your system units are go to **Customise** > **Units Setup** and click on **System Unit Setup.**





In the façade example above, the scene's system units are in centimetres, and the display units are in metres, which explains why 0.01m is used for the offset value. Alternatively you can export the values and wire them to numeric or constant nodes set to output float values. In this case you can use 0.0 - 1.0 to represent UV space.

To randomise UVWs using stepped increments



As well as selecting any value from within a set range, you can also constrain the UVW XForm operator to randomise using only increments of a given step value. For example, you could set the rotation range from 0 to 360 degrees, then by using a step increment of 90, you ensure it only returns values of 0,90,180, and 270. You can do the same thing for Offset, and tile.

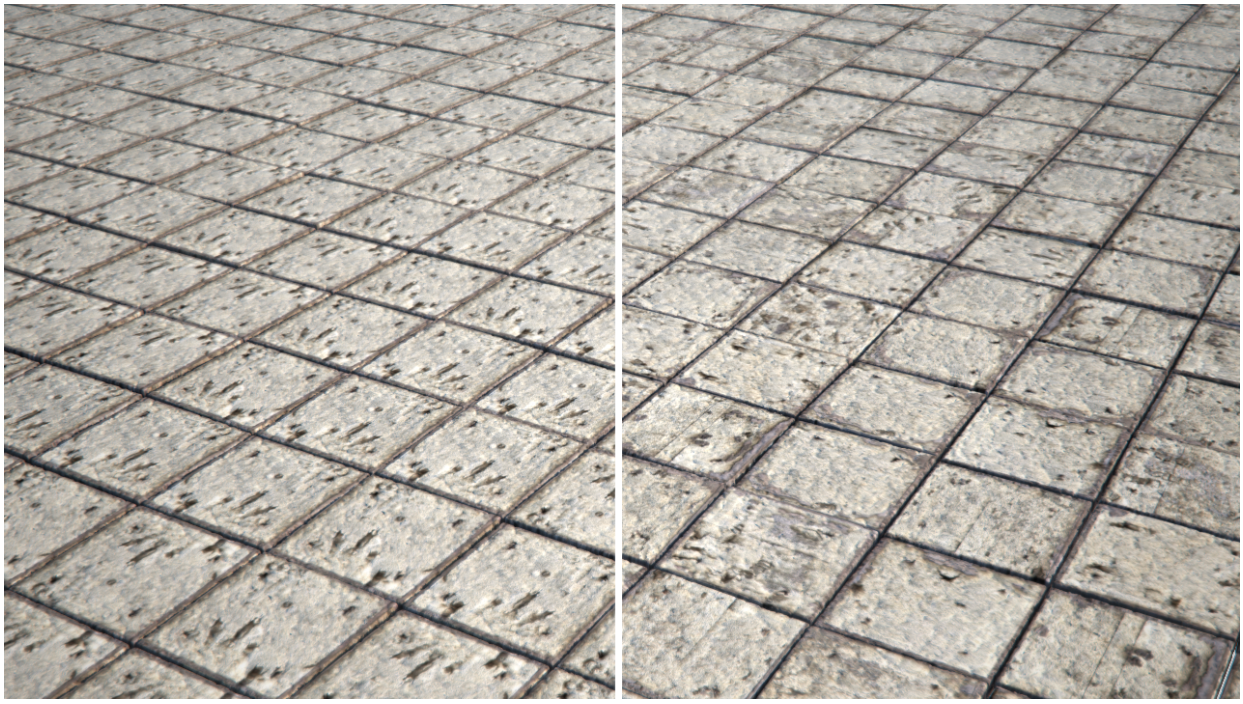
Let's take a look at a typical usage. In this example I have a single tile segment and one texture with 25 tiles in a 5 x 5 grid. Using the old workflow we'd have had to slice this texture up and create a multi sub object material with 25 material IDs. Using this operator we now only need 1 material.

As in the previous examples. You should already have edited the UVs on the segment's source geometry so that only one tile is displayed. Then, to randomise between all 25 tiles.

1. Open chapter_4_stepped_uvw_start.max from the downloads for this guide.

2. Select the RailClone object and open the Style Editor
3. Wire a new **UVW XForm** operator between the existing **Randomise** node and the **Generator's Default input**.
4. Go to the **Random** tab and activate **Offset**
5. Add a **Start** value of **-10mm** and an **End** value of **10mm** to define the U and V ranges (assuming the scene units are set to centimetres, and the display units are in millimetres).
6. Enter a **Step Increment** value, this should be the UV space divide by the number of tiles along a side. In this case that's a UV space of 1 divided by 5 tiles, so we need to enter **0.2**. The random values returned by the operator will be multiples of this figure.
7. We can add further variation by randomising the **Rotation**. Enter a **Start** value of **0** and and **End** value of **360**.
8. Next enter a **Step Increment** value of **90** degrees.

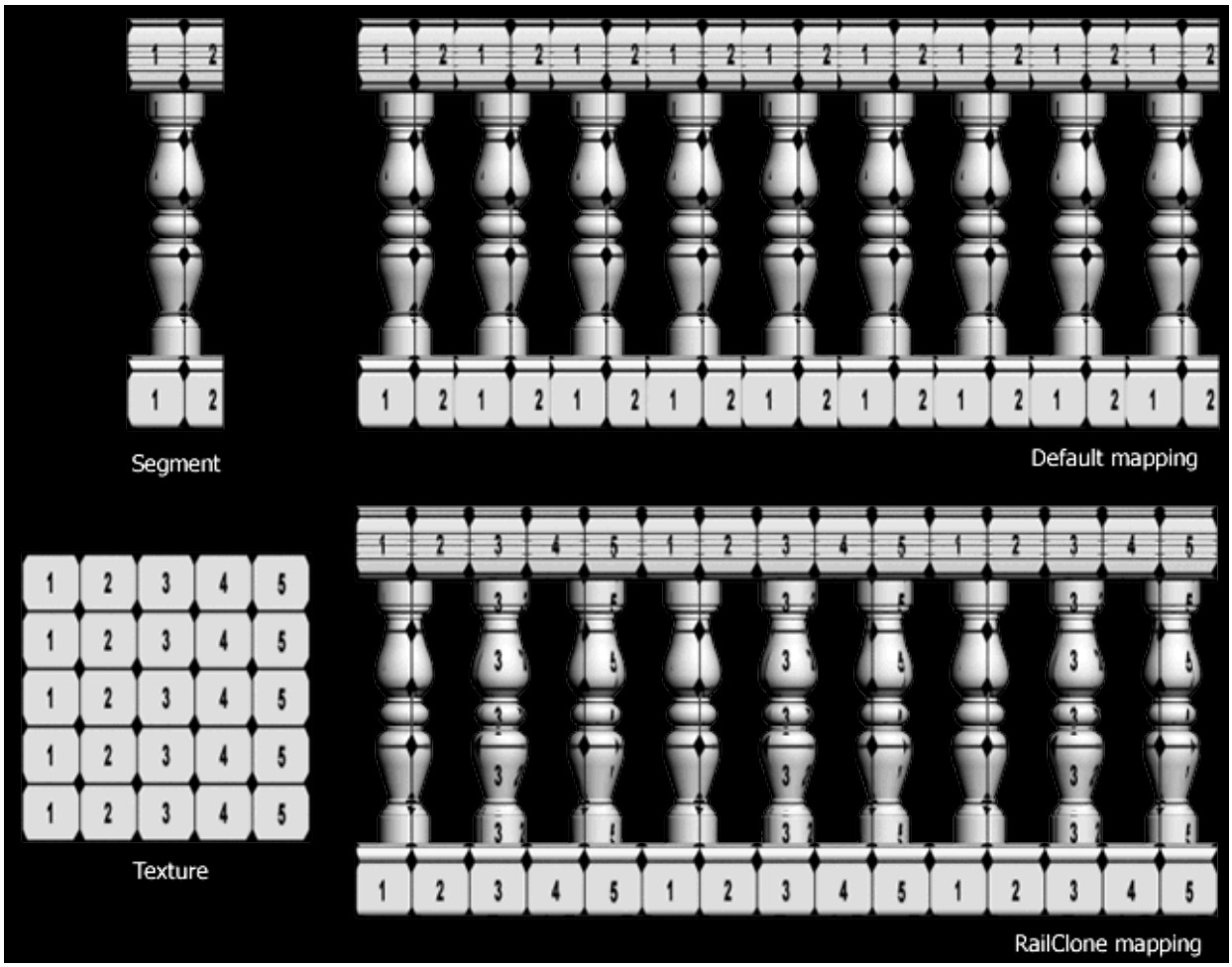
After applying this operator the textures displayed on each segment will now be one of 25 tiles, with 4 possible rotation angles.



4.3 Automatically Box Mapping Segments

RailClone uses regular repeating segments to build objects, but this can result in obvious repetition of textures. To help alleviate this, RailClone allows you to automatically create UVWs that span several adjacent segments. To do this it uses a box mapping algorithm that is able to follow the path so if your segments deform to follow a curve - so will your box mapping! The

diagram below illustrate before and after box mapping a balustrade. You can see in the first example the texture is repeated on each segment, but in the second example the texture continues seamlessly across all the segments

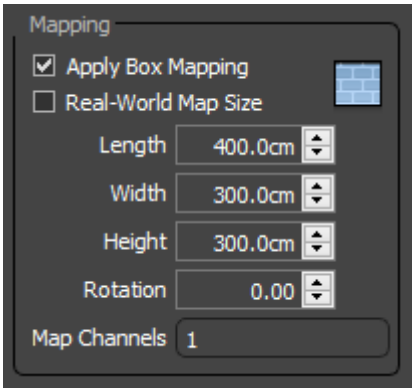


Exercise: Adding Box mapping to segments.

In this exercise we'll box map a simple curved wall. Without box mapping you can very clearly see the texture repetition.

To fix this, follow these steps.

1. Open *chapter_4_box_mapping_start.max* and select the RailClone object called *rc_wall*.
2. Open the Style Editor and select the Segment. From the Properties panel, choose the **Deform** tab and find **Mapping.**
3. Turn on **Apply Box Mapping.**
4. You can either set the size of the box map using the **Length**, **Width** and **Height** settings, or if you are using materials with real world map size settings simply turn this option **On**. In this case we'll set a value of **400cm** for the **Length**, **300cm** for **Width** and **300cm** for the **Height**.



5. You can also use the **Rotation** parameter to align the map correctly on the segment and set the UVW channels you wish to map. In this case we don't need to change the settings and the wall is now mapped. Texture repetition is dramatically reduced.

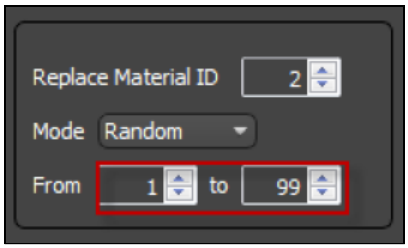
Using Bitmap loaders and randomisers

It is possible to use bitmap loaders and randomisers with RailClone but there are a few things you need to know.

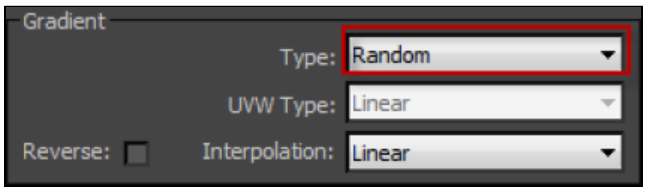
Using multitexture, VRayMultiSubTex, CoronaMultiMap and Mental Ray Multi-Sub Map

These map types works well provided you need **all** the segments in a RailClone object to have randomised materials. They are less successful if you need to add variety to specific segments in a multi-part object. To use them the basic procedure is as follows:

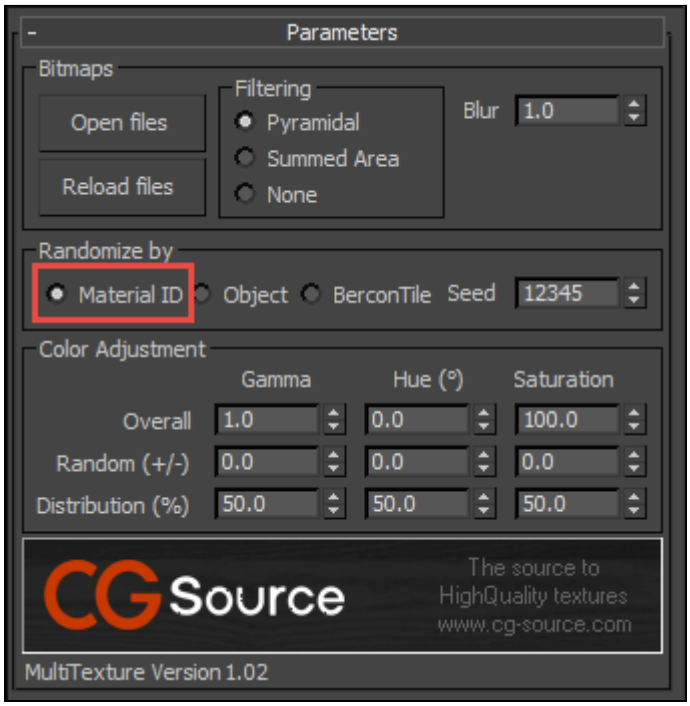
1. Create a style with a new segment to take the randomised textures.
2. Wire the segment to a **Material** node and set it to **Randomise**. The **Range** chosen will determine the number of variations.



3. Create a new material and apply a Bercon Gradient or Multi-Texture map. For Bercon Gradient set the **Type** to **Random** and make sure the **Randomise by Type** is set to **Material ID**



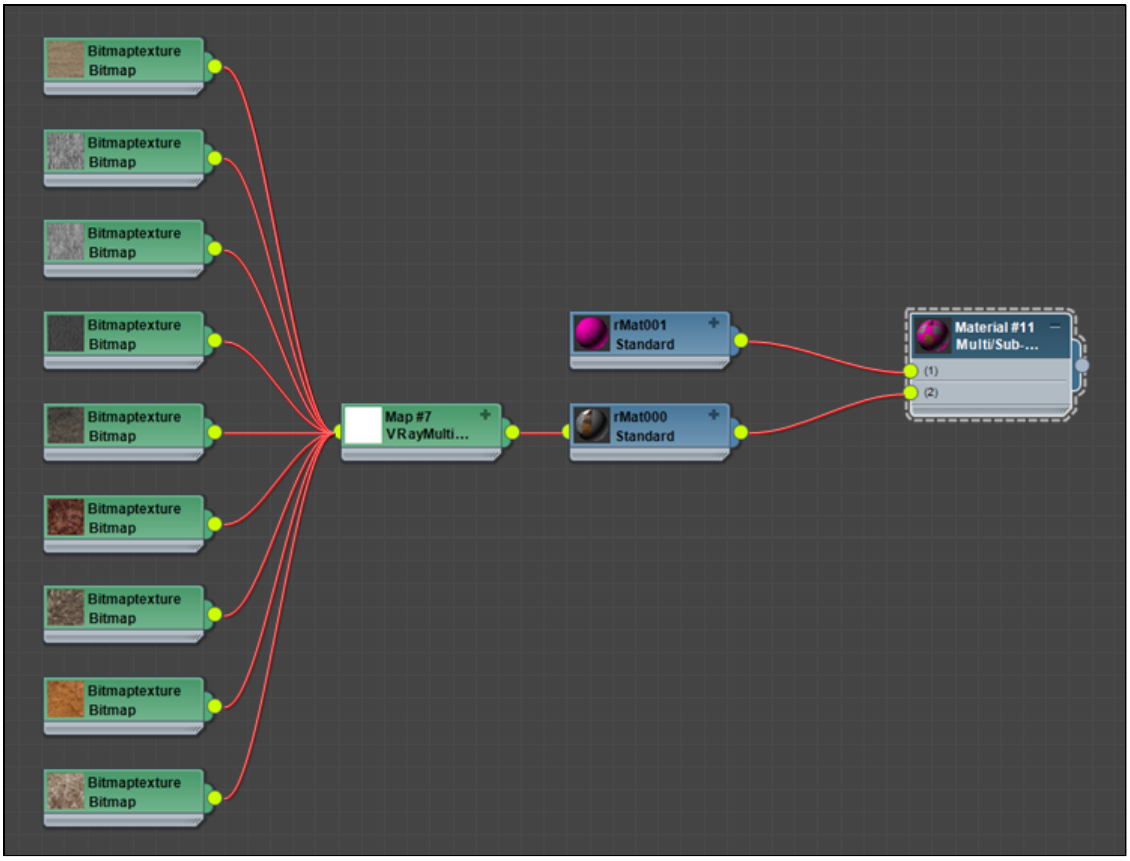
For MultiTexture, set **Randomise By type** to **Material ID** or the equivalent setting in your chosen bitmap loader.



4. That's it! The textures should now be randomised.

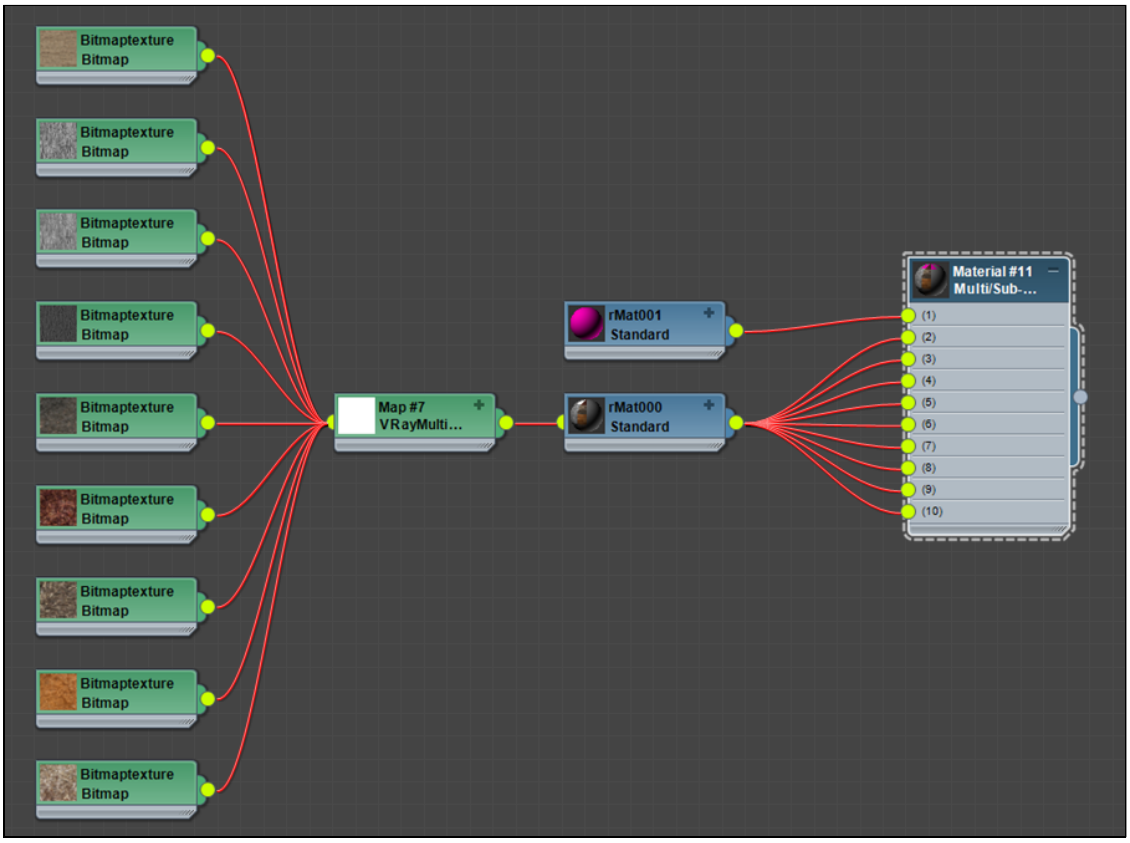
To randomise materials on only a few segments in a multi-part object.

If you want to randomise only a few material IDs then the setup is a little bit more involved. The problem is that the multi-sub material loops through the IDs when the face's ID is out of range on the source geometry. Imagine that you have a Style where ID1 is not randomised but ID2 should be randomised using a bitmap loader, it would be tempting to set it up like this:



Unfortunately this won't work, if the face id is 3,5,7,9 ... you will end up with the material in ID slot 1 instead of 2

The solution therefore is to make sure that the multi-sub texture's number of IDs is the same as the randomisation range of the material operator. So in this example we have 9 textures, you would need to create a Mult-Sub material with the non-randomised material in ID1 and the randomised material with the bitmap loader instanced into ID slots 2-10.

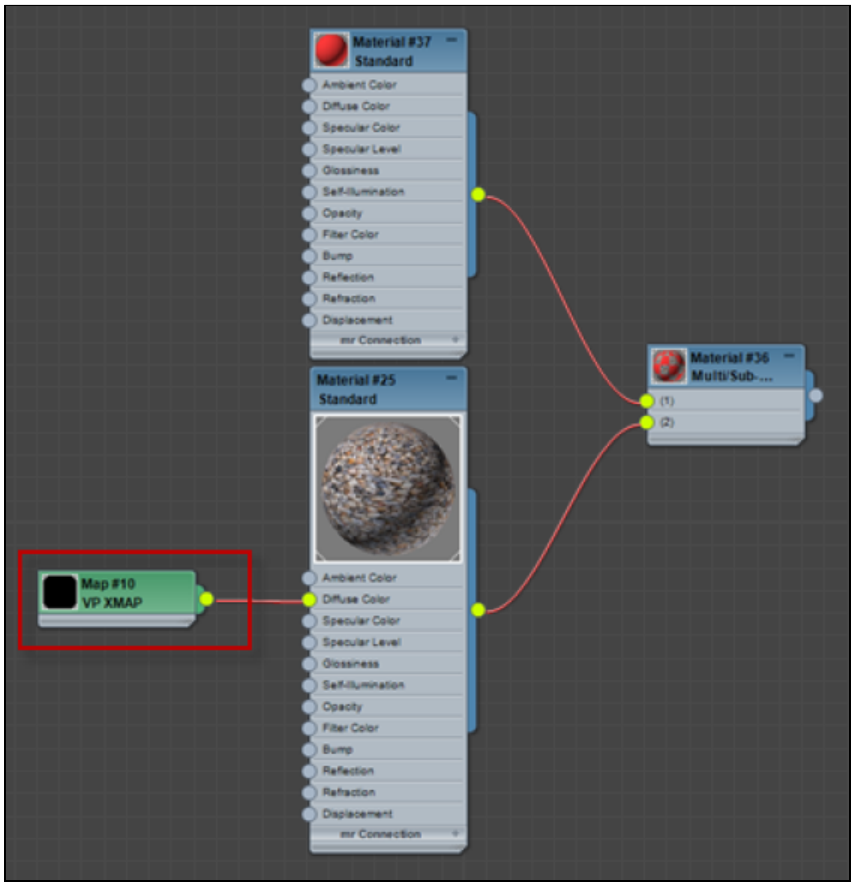


We realise that instancing a material into a large number of multi-sub texture slots can get tedious so there is an unofficial script that can help automate the process for you. You can get it on our forum here.

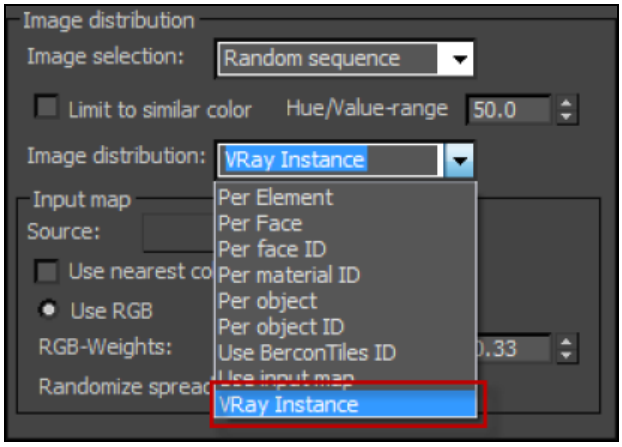
Using Crossmap

Alternatively you can use VizPark's Crossmap quite successfully with RailClone. This is very easy to use.

1. Use **Material IDs** and create a multi/sub-object material for each segment as normal

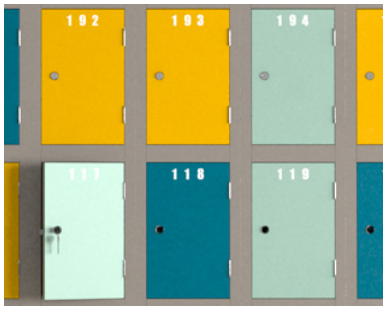


1. Then use the **CrossMap** plugin for the material you wish to randomise ensuring you set the **Image Distribution** settings to **V-Ray Instance**.



2. Despite the name, this seems to work for Mental Ray too!

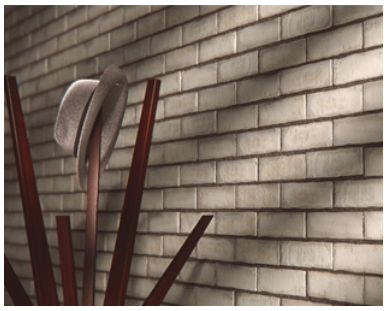
4.4 Related Tutorials



creating_number_sequences

Creating Number Sequences

For some styles it is necessary to create numbers that increment for each segment. In this Tips and Tricks episode we look at two ways to create sequences of numbers. The first example illustrates how to create a series of numbers by cycling through geometry. In the second part we manipulate a segment's material IDs using expressions to create numbers using multi-sub object materials.



mapping_tiles_and_bricks

Mapping Tiles and Bricks

In this advanced exercise we look at a time saving graph-snippet that uses UVW XForm modifiers for automatically mapping segments and randomizing them with individual tiles or bricks from textures.

create_a_seaside_promenade

Seaside Promenade

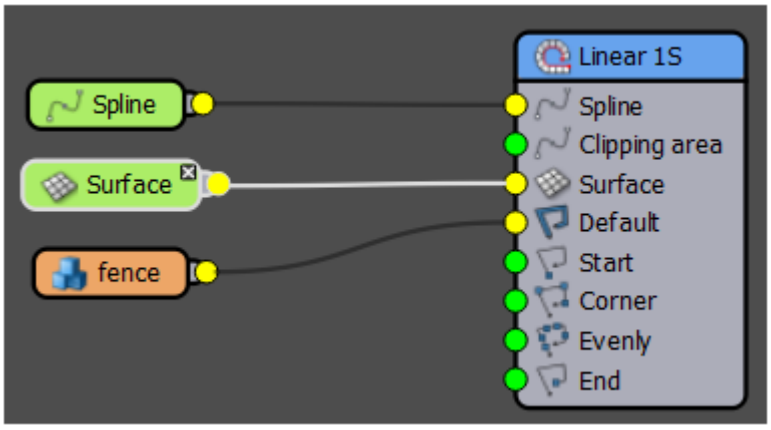
This tutorial demonstrates how to use box mapping to create a seamless stone balustrade.

Documentation

For more information please see the Segments, UVW ZForm, and Material operator sections of the online documentation.

5 5 - How to use Surfaces

If you want a RailClone style to follow the contours of terrain, then RailClone has got you covered with the **Surface** object type. This powerful feature has pleasingly few settings, but can save a lot of time spent creating splines to follow a landscape. To use it, all you need to do is add a Surface node to your graph, pick your terrain, and wire it to the Generator's Surface input. Once this is done, all the segments used in the style will be projected along the RailClone objects **local Z axis** until they meet the surface geometry.



Like all of RailClone's features this is fully parametric and interactive. Editing either the surface's source geometry or the spline path will result in instant feedback in the viewports, making it invaluable for a fast iterative workflow.

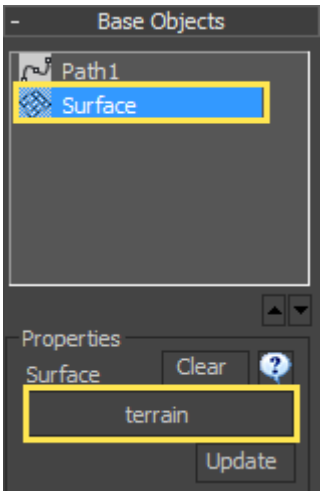
In the following exercises we'll learn how to use surfaces with L1S and A2S arrays, as well as sharing some tips to make working with this powerful feature as simple as possible.

5.1 Exercise: Using Surfaces with an L1S Generator

Using a surface with the L1S generator is as easy as attaching a single node. Let's take a look at how we can add a surface to a brick wall so that it smoothing follow the terrain.

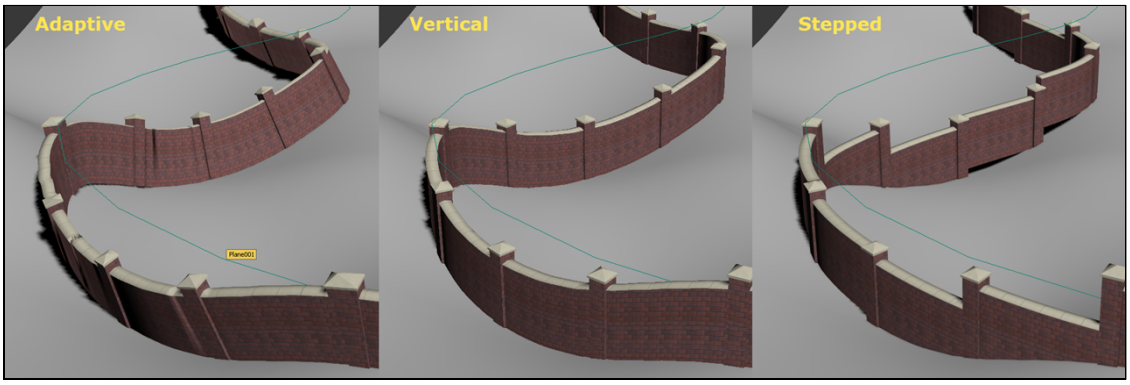
1. Open *chapter_5_surfaces_1.max* from the downloads for this guide.
2. Select the existing brick wall RailClone object and open the Style Editor.
3. Drag a new **Surface** object to the Construction view.
4. Wire the **Surfaces** node to the **Surface input** of the L1S Generator.

5. From the **Base Objects** rollout in the Modify Panel, select the surface from the list then click the surface picker button and select the object called *terrain* from the scene.



6. For the two segment's change the **Properties** > **Deform** Mode to **Vertical**. This will stop the wall from banking as it follows the projected spline path.
7. Finally, to make the deformation smoother, go to the **Style Rollout** in the Modify Panel and increase the number of **Curve Steps** to **50**.

The Segment's **Properties** > **Deform Mode** will still influence deformation when using a surface, and as the spline is changed when projecting onto an undulating surface you may see some behaviour that wasn't apparent when the spline was flat. In particular when using Adaptive mode, you may notice the segments are no longer upright. This is due to the segments banking as they follow the uneven spline. To fix this you can change the mode to **Vertical** or **Stepped.**



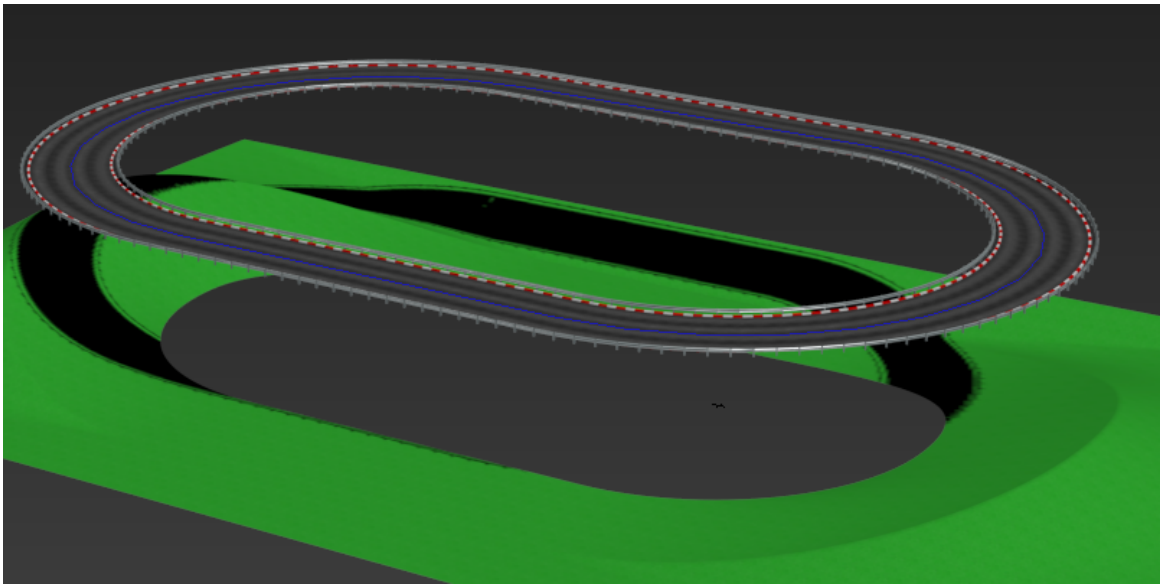
Segments are projected along the RailClone object's local Z axis until they hits a surface. Any faces in the surface object which are facing away from this axis will be ignored. For this reason it is also important that the face normals are not inverted as can often be the case with imported geometry. If the surface is not behaving as you'd expect, try adding a Normal modifier and turn on **Invert Normals**.

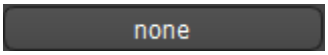
5.2 Align Y with Surface (L1S Only)

As well as deforming along the Z axis, when using an L1S generator segments can also conform to the surface on the Y Axis. This is enabled individually for each Segment and can be found in **Properties** > **Deform** > **Deform** > **Align Y with Surface**. When the option is off (the default setting), then the segment's remain vertical. If Align Y with surface is turned on, then the segment will rotate so that it is perpendicular to the surface.To see how this works, try the following exercise to project a road and barriers onto a camber.

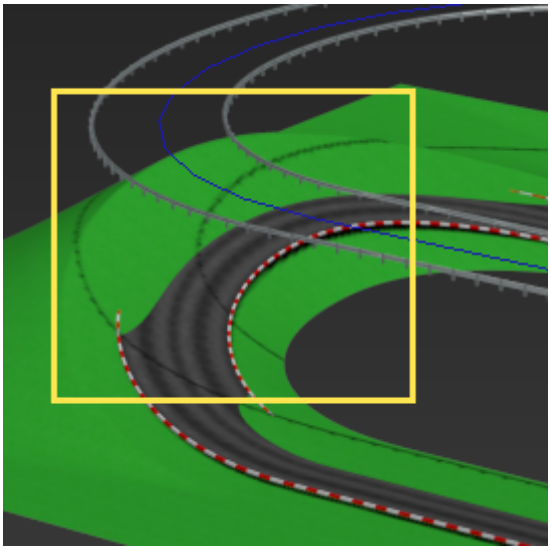
Exercise: Align Y with Surface

1. Open the file named *chapter_5_surfaces_2.max* from the downloads for this guide, you'll see a surface, a road and a barrier.

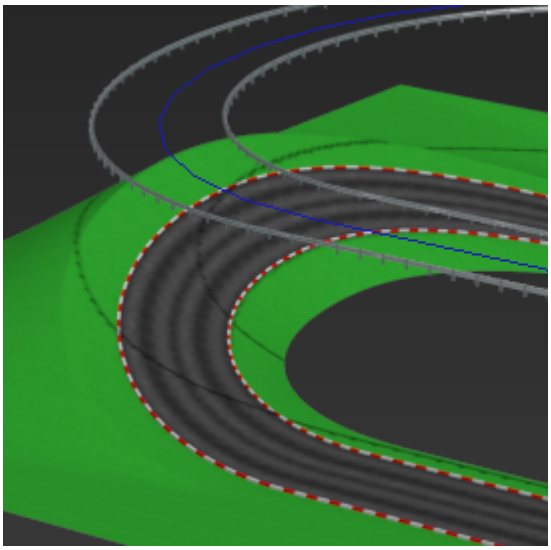


2. Select the Road RailClone object and open the style editor.
3. Wire a new **Surface** node to the Generator's Surface input. Open the **Surface** nodes properties, click 

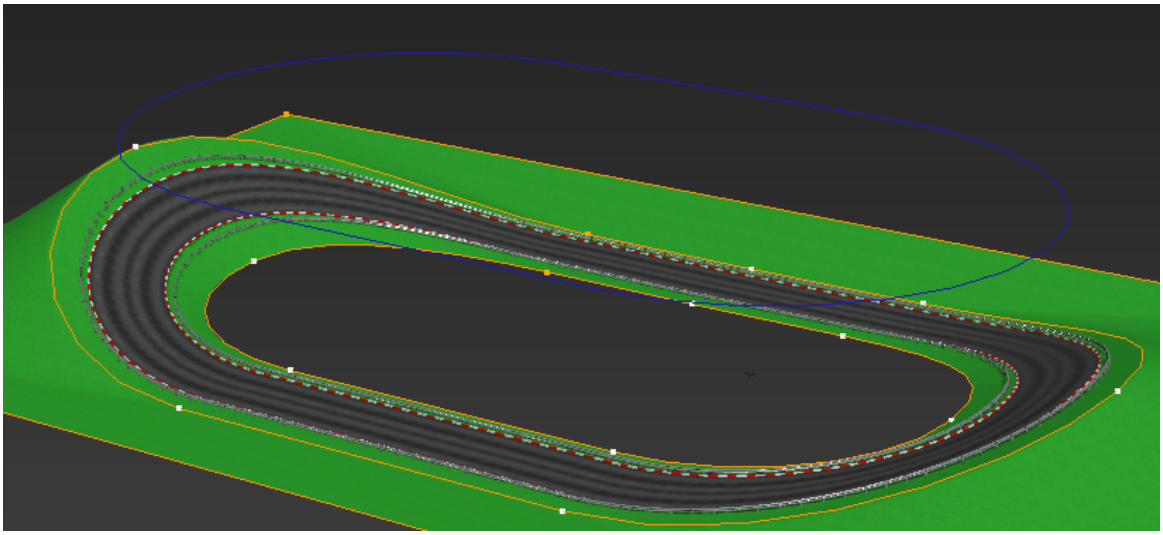
and select the object named *terrain*.The road will now project to the surface, but it will intersect as it's not following the geometry on the Y axis.



4. Select the Segment node and go to **Properties** > **Default** and activate **Align Y with Surface.** The road will now conform to the surface on the Y axis.



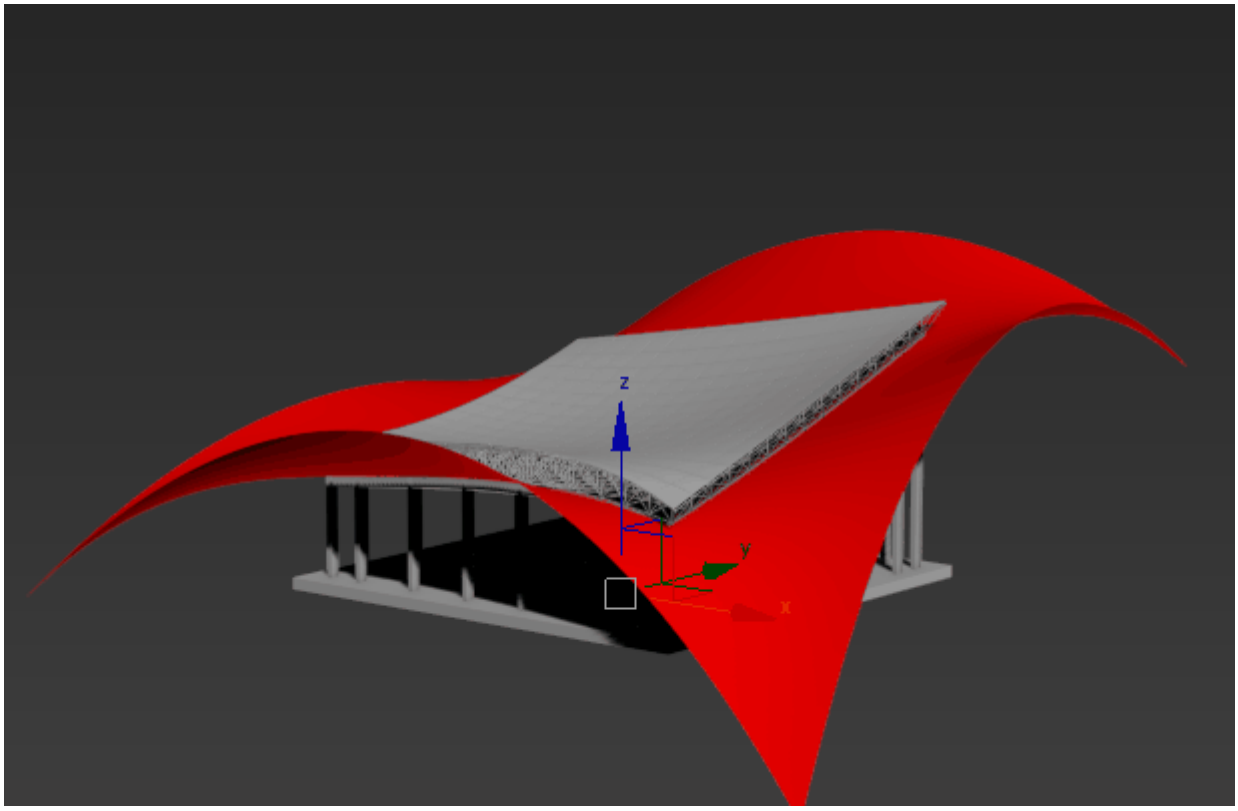
5. Repeat the above steps for the *Barrier* RailClone object. When this is done both the barrier and the road will follow the terrain and orientate so that the segment's Z axis is perpendicular to the surface. Both the spline and surface can be edited interactively to give you instant feedback.



5.3 Using Surfaces with an A2S Generator

In the examples above we used a surface with the L1S generator, but it's also possible to use the A2S generator to conform a two dimensional array to a surface.

1. Open the file named *chapter_5_space_frame_start*.
2. The RailClone object in this scene creates a space frame using an A2S array. Let's deform it to follow a surface. Go to the **Styles** rollout and open the **Style Editor**.
3. Drag a new **Surface** object from the Items List to the Construction View and wire it to **all 4** generators. Open the Surface node's properties, click none and select the object named *deformer*. The space frame will now deform to follow the surface!



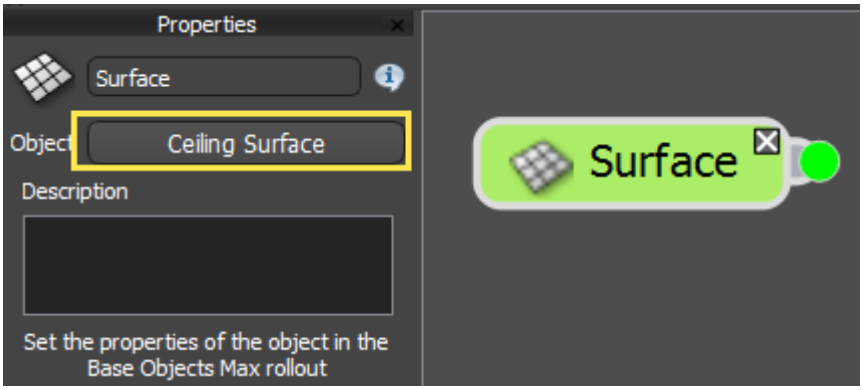
In this example the surface is only used to deform the RailClone object and we don't want to see it in the final image. To remove it from renders, right click on the surface and select **Properties.** From the **Object Properties** options, turn off **Renderable.**

In the example above you'll notice that the geometry deforms automatically on the X and Y axes to follow the curvature of the surface. When using an A2S array the Align Y with Surface checkbox has no effect. However you do have the ability to change the **Deform** mode to create a variety of different effects. In the next example we'll look at using a surface with segment's set to **Deform** > **Step** mode to create a ceiling.

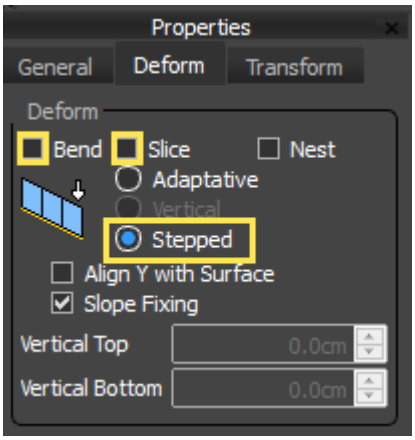
Exercise: Using Surfaces with Stepped Mode

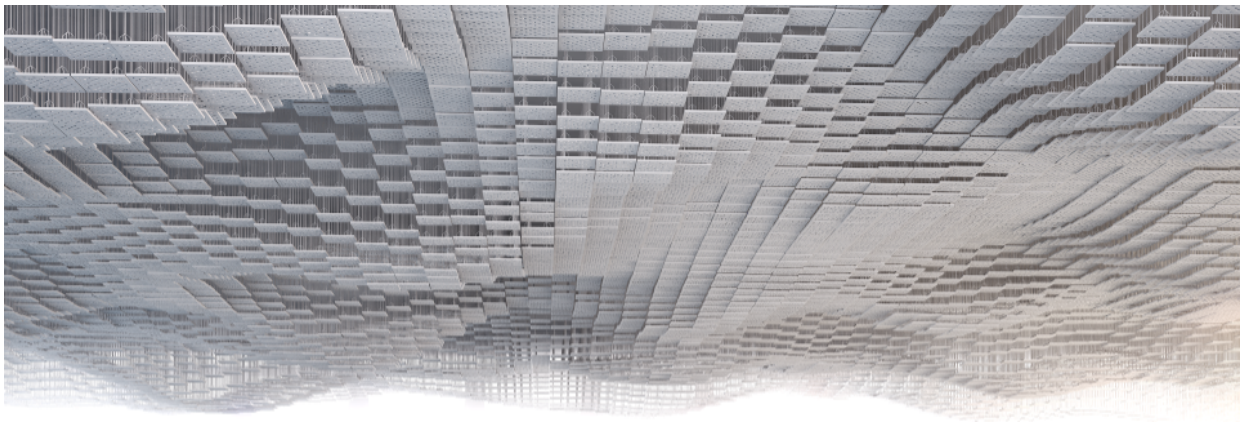
In this example we'll look at how surface mode could be used to create an undulating suspended ceiling. In this case we don't want the geometry to deform to follow the surface. This will ensure that the style looks correct, but also that instancing is maintained for maximum rendering efficiency.

1. Open *chapter_5_suspended_ceiling_start.max* from the downloads for this guide. The scene already contains a style with two A2S array, one for the tiles and another for the ceiling supports.
2. To add the surface, create a new Surface node and select *Ceiling Surface* as the source.



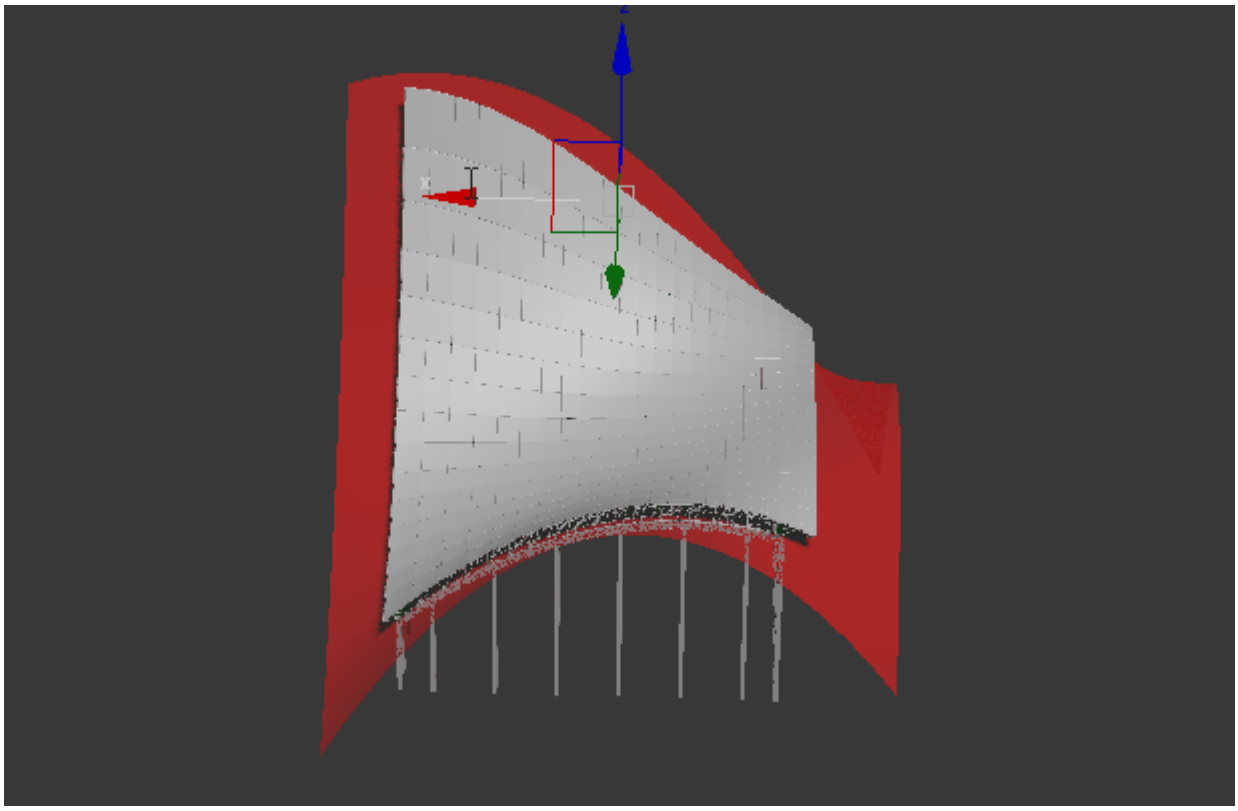
3. Wire the Surface node to the Surface input of both generators
4. The segments in both generators will be deformed to follow the curvature of the surface. To fix this we need to change the Segment's properties...
5. For each segment change **Deform Mode** from Adaptive to **Stepped**. As the segments are not longer deformed they will remain instances and us substantially less memory. If you wanted to be thorough and force all the segments to render as instances you could also turn off **Bend** and **Slice**.



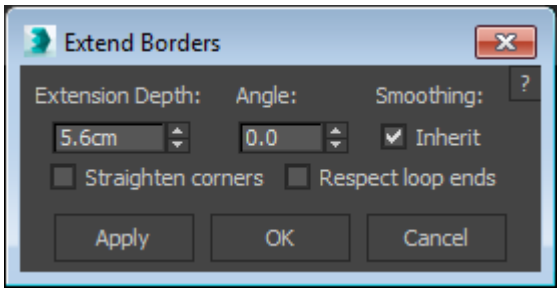


5.4 Extending Boundaries

It is important when using surfaces that the array is completely enclosed within the boundary of the surface. Any geometry which falls outside of the boundary will be deformed incorrectly, as you can see in the example below .

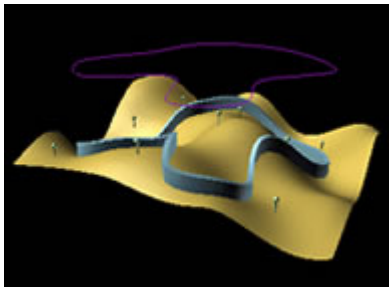


You may also get deformation if the boundary of the spline and the boundary of the RailClone object are exactly the same. Surfaces work best when they are slightly larger that the RailClone object that they deform. This situation can often occur if you extract a clipping spline to use in RailClone from the surface's perimeter. If you do this then you should also extrude the surface's open edges slightly so that they are larger than the clipping spline. For complex surfaces getting a decent extrude that continues the direction of the adjacent polygons can be tricky, but we find the Extend Borders script by Racoon Artworks does a pretty good job.



In this chapter we looked at how you can use Surfaces to deform an array and its segments. In the next chapter we'll look at how segments can be sliced using closed splines with RailClone's clipping paths feature.

5.5 Related Information



Glue

Glue Plugin

The Glue plugin is one of our free tools for conforming splines to a surface. In most cases this isn't required for working with RailClone which has it's own interactive algorithm for creating the same effect. However there are a couple of instances where you may still wish to manually create a spline to follow a landscape. This is usually required if you need an A2S array which has been rotated using the Generator's X-Rotate feature to follow a surface. At present RailClone's surface feature does not work with this configuration so as a workaround you can manually create the spline using the Glue plugin

Documentation

For more information please see the 1D arrays - Generator L1S, 2D arrays - Generator A2S, Segments and Surface sections of the online documentation.

6 6 - How to use Clipping Splines

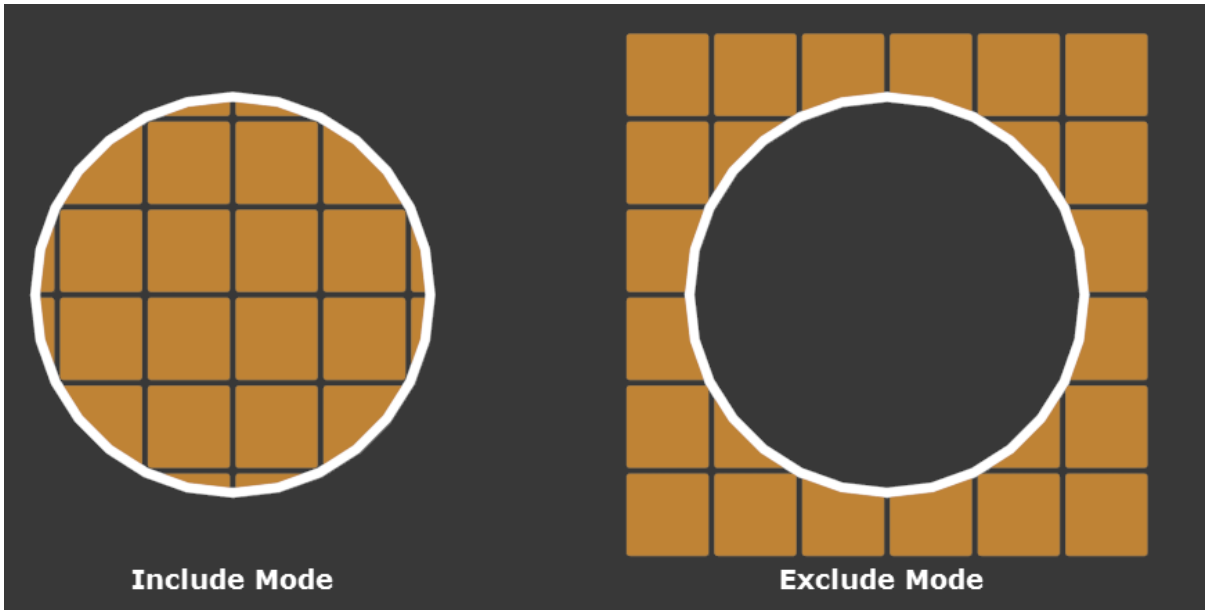
Clipping allows you to use a closed spline to slice the geometry of a RailClone object. RailClone's clipping is considerably more efficient than using 3DS Max's boolean operations as it only computes segments that are intersected by the clipping spline and it is able to maintain instancing for non-sliced segments. To use clipping splines there are 5 rules that are helpful to avoid encountering problems.

Rule 1 - Clipping splines must be closed, open splines have no effect.
Rule 2 - Sliced segments are no longer instanced, however other segments retain all the benefits of instanced geometry. This can be a consideration if your spline is slicing a large number of very high polygon segments as you can end up with high polygon counts if you're not careful.
Rule 3 - The **Projection** axis relates to the local pivot system of the RailClone object - not the spline's or world's axes.
Rule 4 - Clipping splines cannot be used simultaneously as include and exclude areas in the same style. To do this you need to add two different Spline nodes and pick the same spline twice.
Rule 5 - When using **Extend X/Y Size to area**, **Generator** > **Limit by Material ID** is disabled.

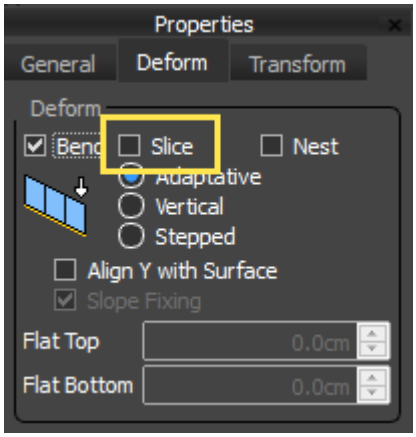
6.1 Include and Exclude geometry

In it's simplest Mode, clipping splines allow you to use a boolean operation to include or exclude parts of the array. In **Include** mode, segments that fall entirely outside the clipping area are removed, segments that are entirely inside the clipping area are retained, and any segment that the spline crosses will be sliced and the section outside is removed.

The opposite is true if you set the clipping operation to **Exclude** mode. When this is selected, segments that falls entirely outside the clipping area are retained, segments that are entirely inside the clipping area are removed, and any segment that the spline crosses will be sliced and the area inside is removed.

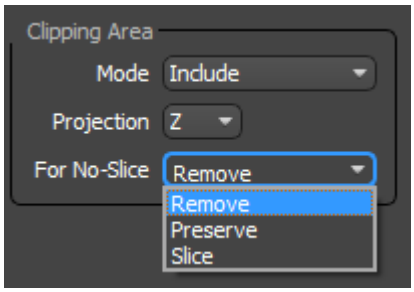


These two modes can also be controlled further using the **Segment's Properties** > **Default** > **Slice** toggle in conjunction with the generator's **For No-Slice** options.



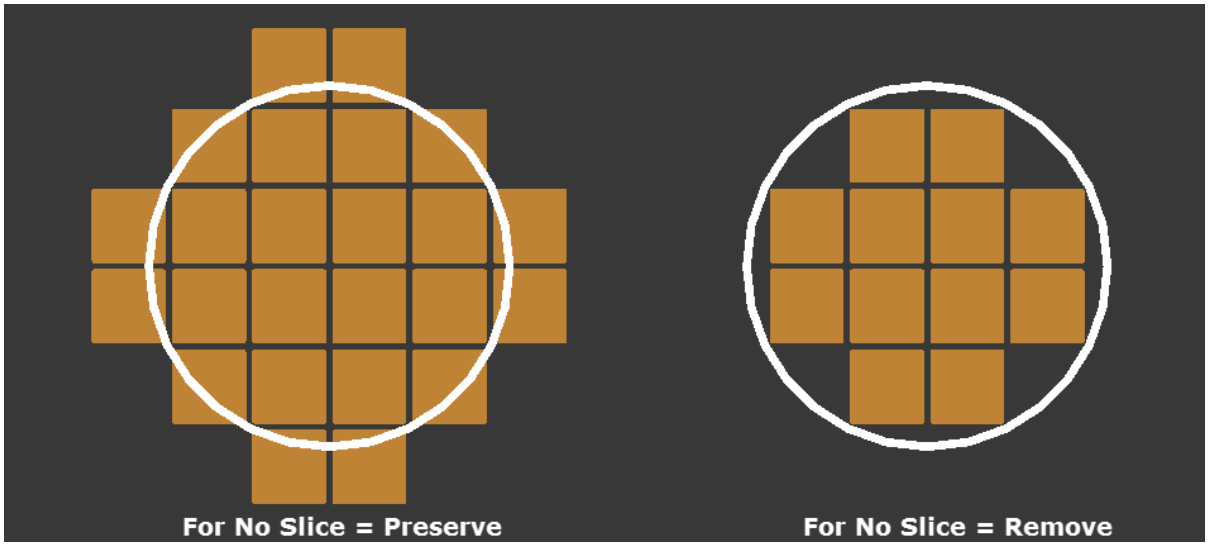
The Segment's Slice On/Off option box

For each segment you can choose to turn on or off the ability to slice the geometry.

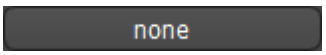


The Generator's For No Slice options

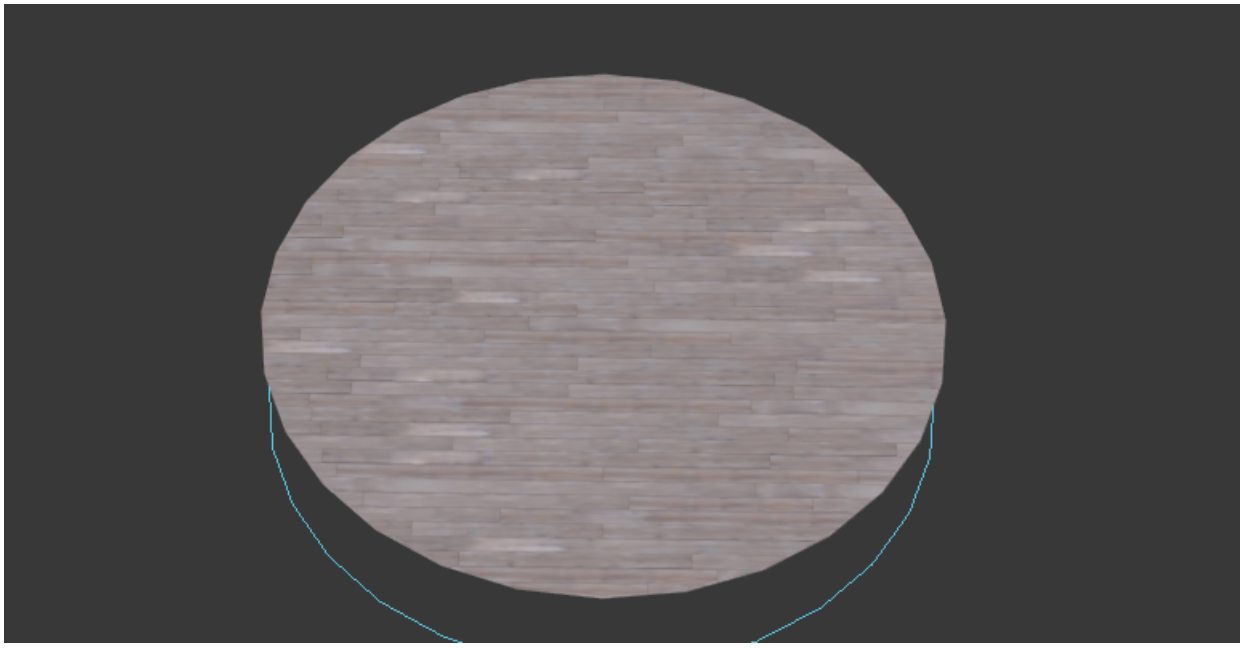
Then, when a segment with slice turned off intersects with a clipping spline you have 3 options. The segment can either be removed completely, preserved, or the slice toggle can be overridden in which case the geometry will be sliced irrespective of the segment's settings. The diagram below illustrate **Preserve** and **Remove** mode on segments with Slice turned **Off**.



Exercise: Using Clipping to Include geometry

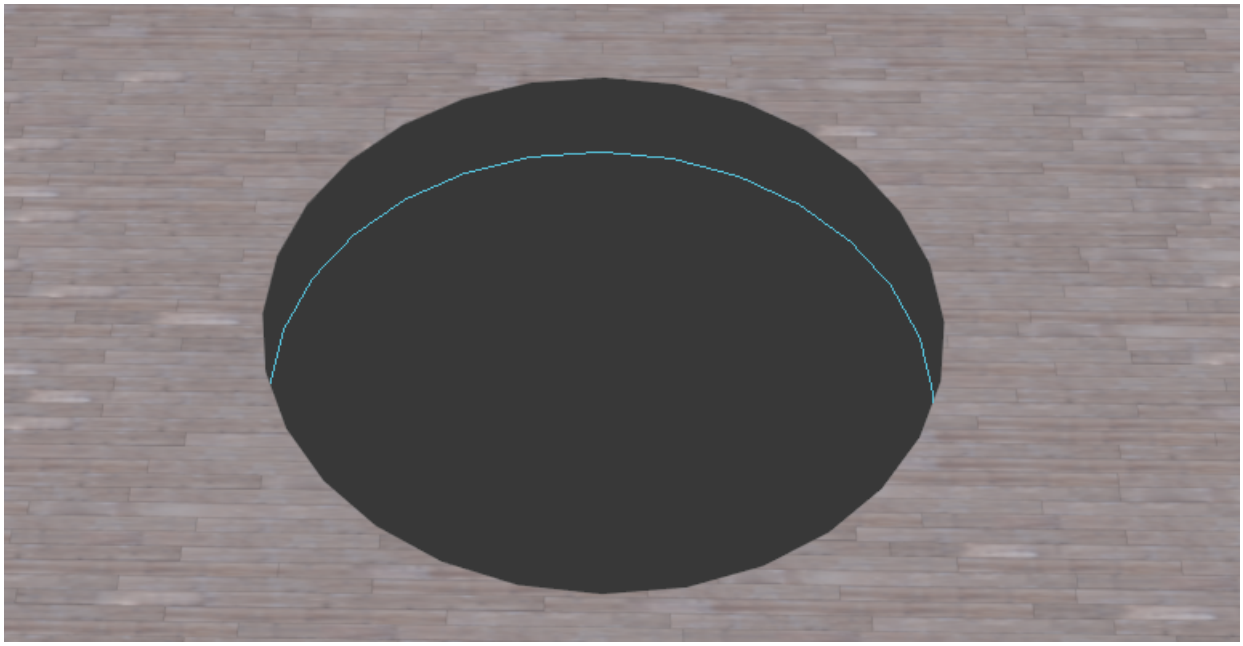
1. Open the file called *chapter_6_deck_start* from the downloads for this guide.
2. Select the *rc_decking* RailClone object. This is a simple style that uses a couple of board segments to create a wooden deck driven by 2 splines to define the size on the X/Y axis. There are also two additional generators that are currently turned off. These will be used in the following exercises, you can ignore them for the minute.
3. Drag a new **Spline** object from the items list to the construction view. Wire this the Generator's **Clipping Area** input.
4. Select the new Spline node, go to Properties and click 

to select a spline from the scene. Pick *circle_clipping_spline*, you'll find it's under the decking.
5. Select the generator and go to the **Properties**. In the **General** tab you will find the Clipping settings. Change the mode to **Include** (if it isn't already) and the deck will only appears inside the circular spline.



Exercise: Using Clipping to Exclude Geometry

To set the style up to use the spline to cut through the geometry and leave the outside area intact you would follow exactly the same procedure as described above except in the final step change the mode to **Exclude** and you'll see that the circular splines cuts a hole through the deck!



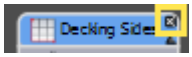
Exercise: Re-use the clipping spline.

Clipping splines in RailClone are just like any other object, they can be easily re-used in various ways to create sophisticated interrelated rules-sets.

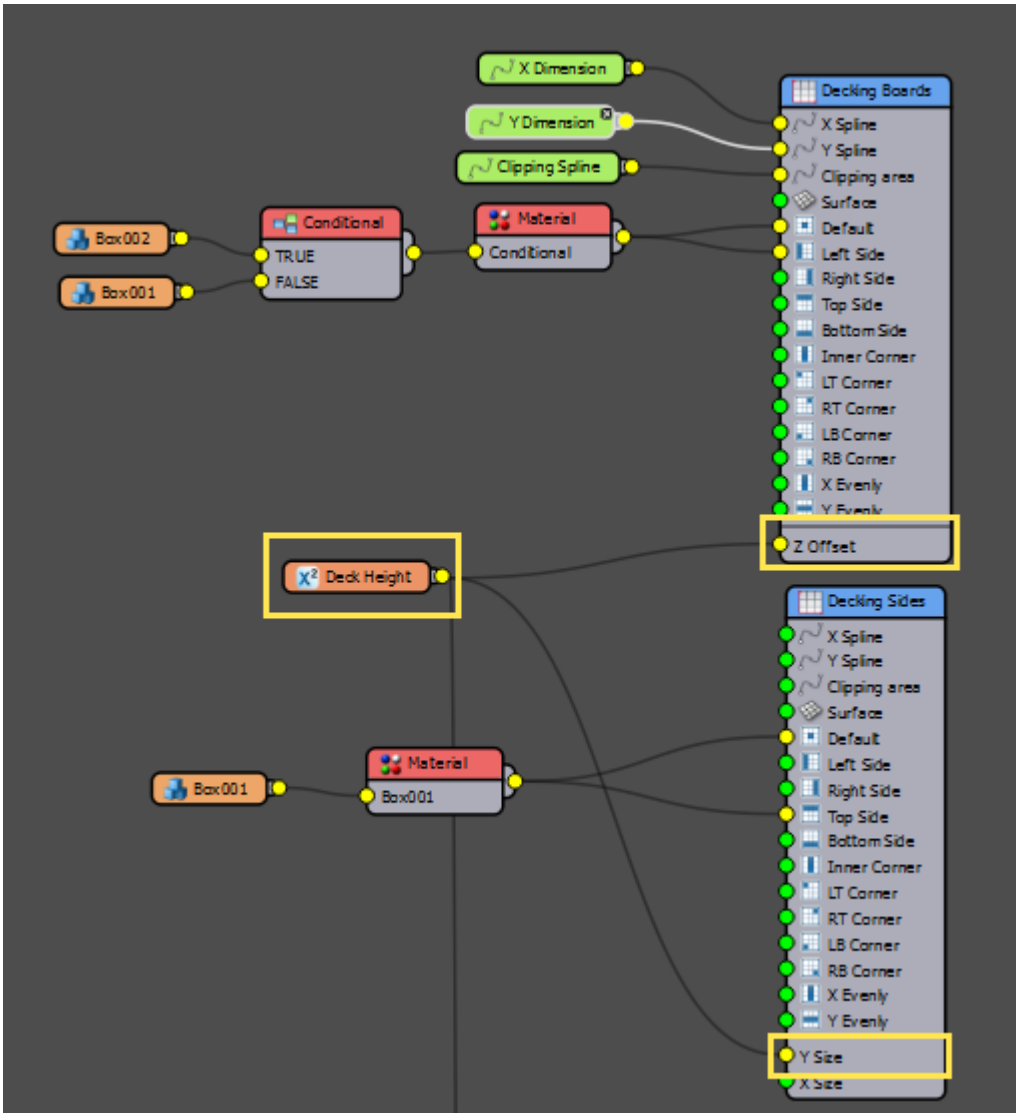
You can use a clipping spline with multiple generators as long as the same mode is used for each generator - it isn't currently possible to use the same spline node to exclude geometry in one generator, and include it in another. If you need to do this you will need to create two spline nodes - one for excludes and another for includes. These can however both use the same spline as a source object.

It's also possible to reuse a clipping spline to define the X or Y axis of another generator. This is extremely useful if you need to create a border around a clipped area. In the following example we'll illustrate how the decking style uses a Second A2S generator to create a vertical edging, and an L1S generator to add a balustrade around the perimeter of the clipped area. Both of these use the same spline node, so if you edit the path's position or shape - all three generators will update automatically.

1. Starting from the end of the previous exercise, re-open the Style Editor. In the scene are two other generators, one for the edging and another for the balustrade.

2. Turn on the generator called *Decking Side* by clicking on the X in the top right hand corner of the node 

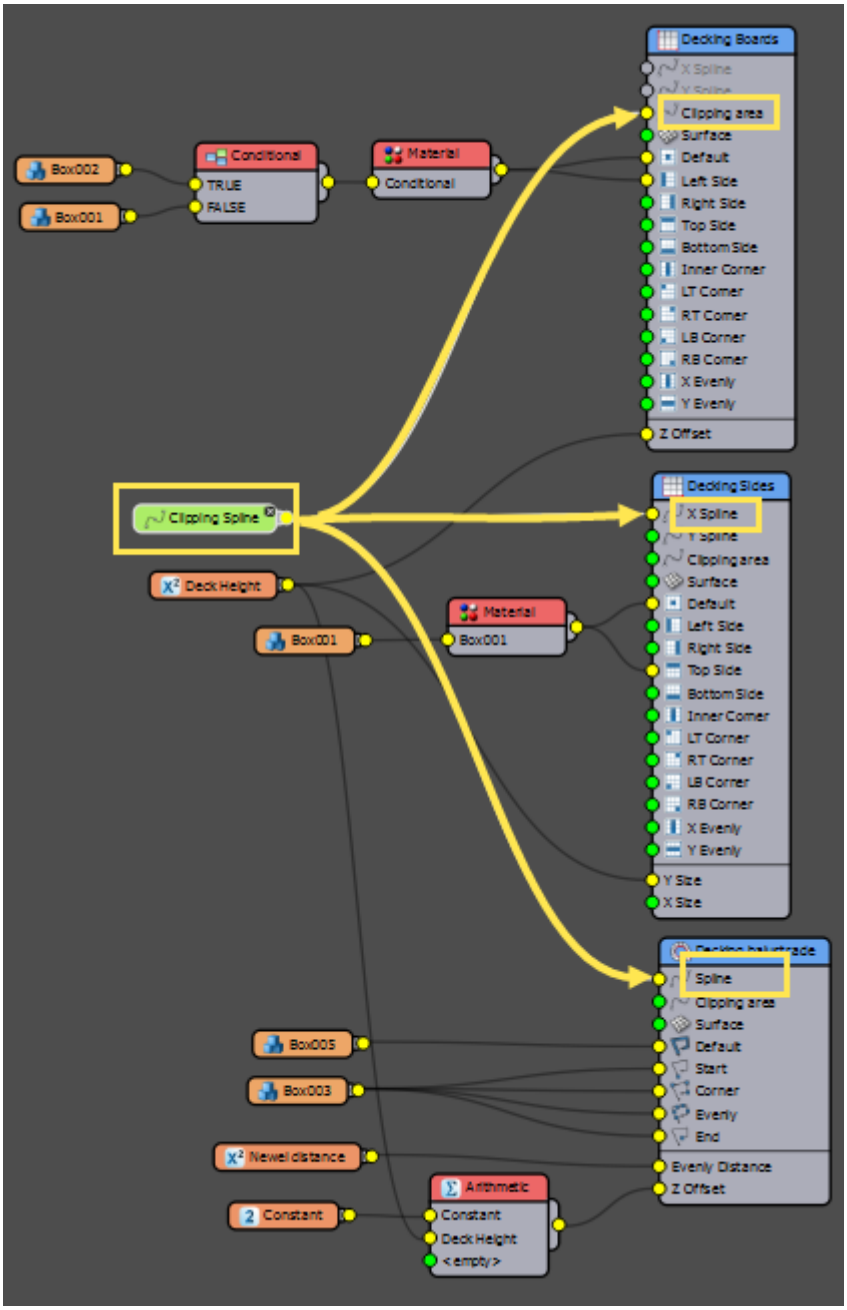
. Note that this Generator's Y Size and the Decking Generator's Z offset are both wired to the same **Numeric** node. This means that as you raise the decking object, the height of the sides will increase to fill in the space between the deck and the floor.



3. To set the path for the Generator's X axis, wire the clipping spline the the X Spline input.

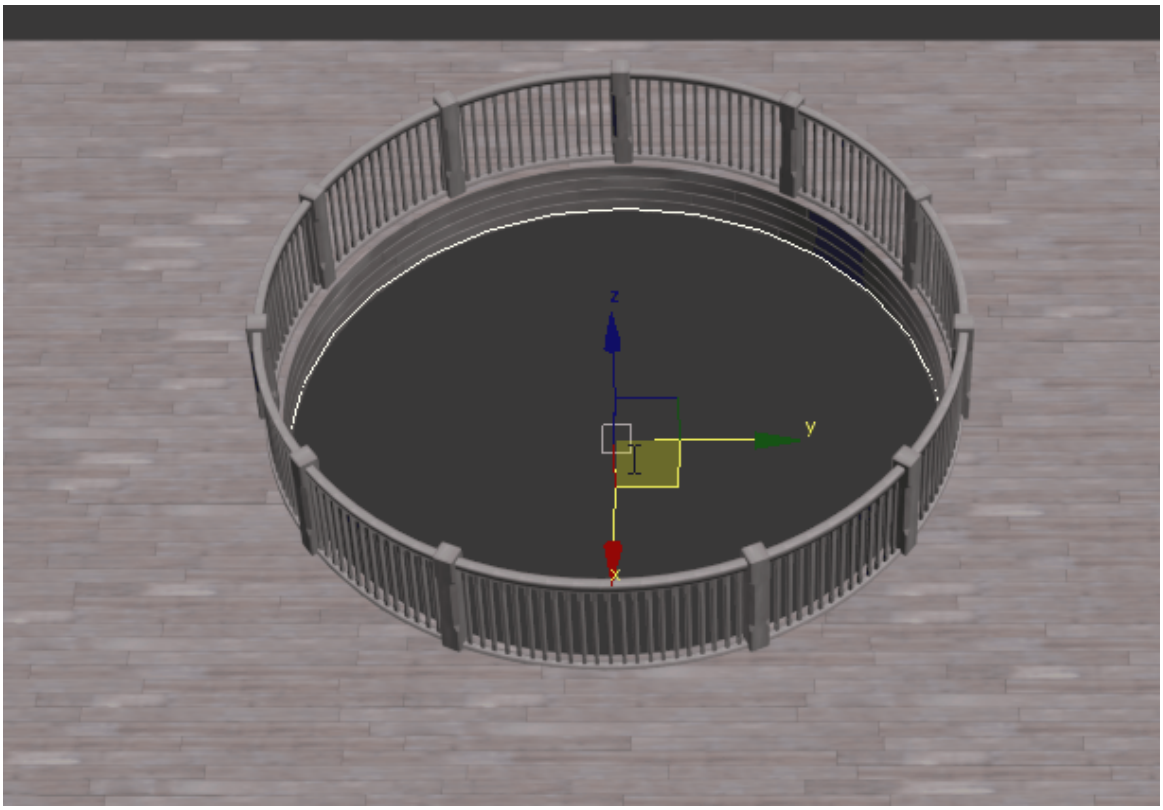
4.

Now we'll do the same thing for the balustrades. Turn on the 3rd generator and wire the Clipping spline to the generator's spline input .



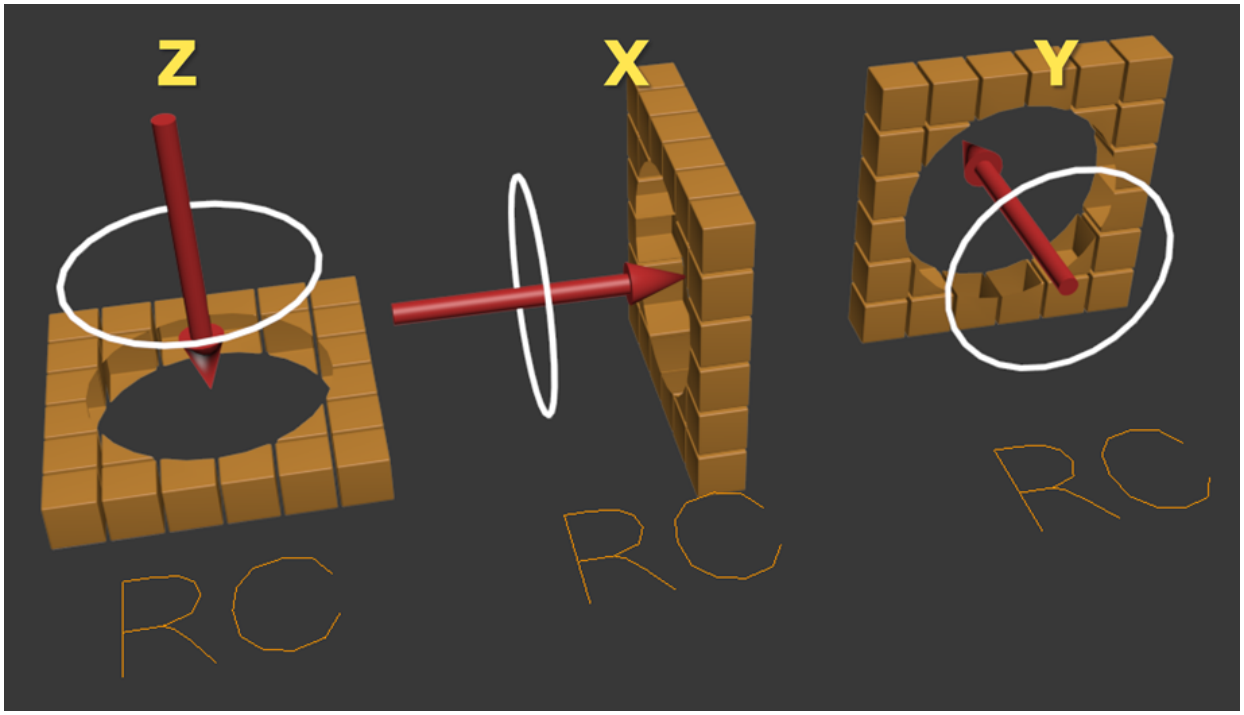
These three generators are now driven by the same two inputs. If you move or reshape the spline or adjust the Deck Height parameter, you'll see that all of the generators are

updated.



6.2 Changing Projection Axis

In the previous example, the clipping spline is projected along the RailClone object's local Z axis. This is the default mode but it is also possible to change the axis to project on the X or Y axes.

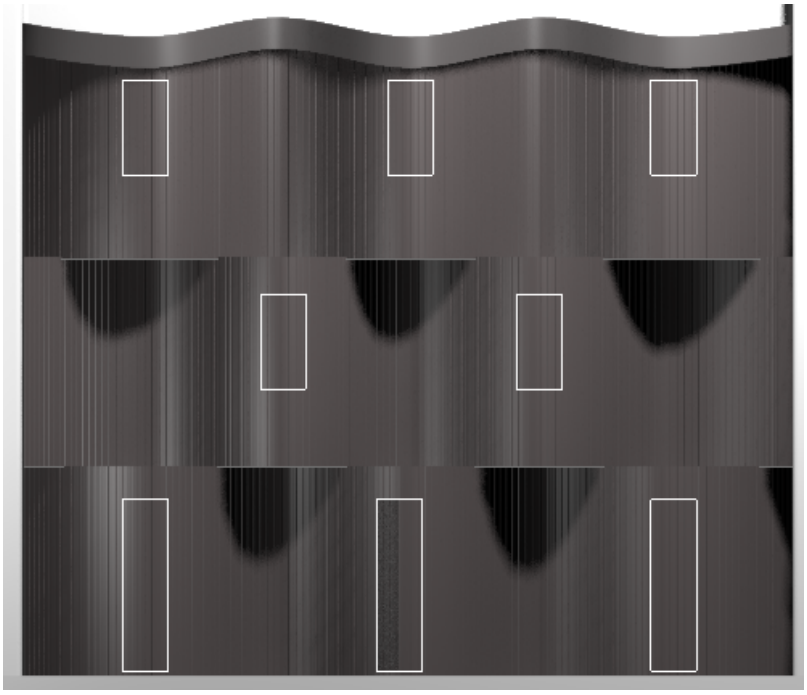


This is ideal when you need to clip a vertical style like a wall or façade.In the following example we'll change the projection axis to clip windows from a cladding style.

Exercise: Clipping holes in a Vertical Style

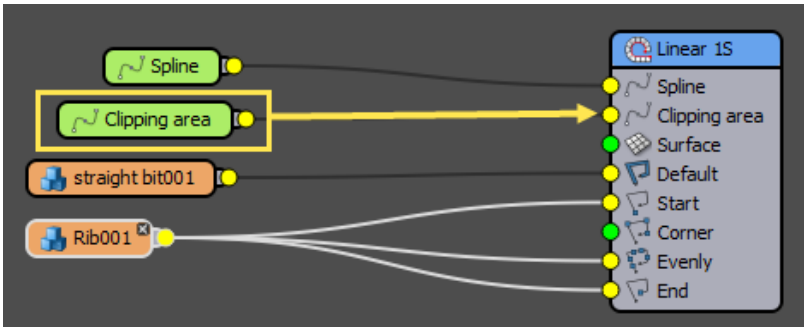
1. Open ***chapter_6_clipping_axis*_start** from the downloads for this guide.

2. In this scene you'll find a vertically clad wall. In front of this is a spline consisting of 8 rectangles that describe the size and location of window apertures. We can use this spline to cut holes through the cladding for the windows (which would be added as a separate object).

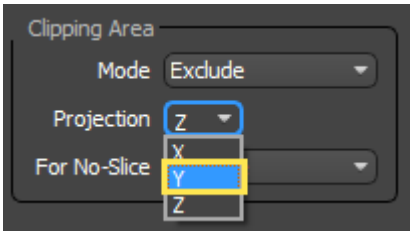


3. Select the *rc_cladding* RailClone object and open the Style Editor. Create a new Spline node, go to the **Properties** panel and click none to pick *window spline* from the scene.

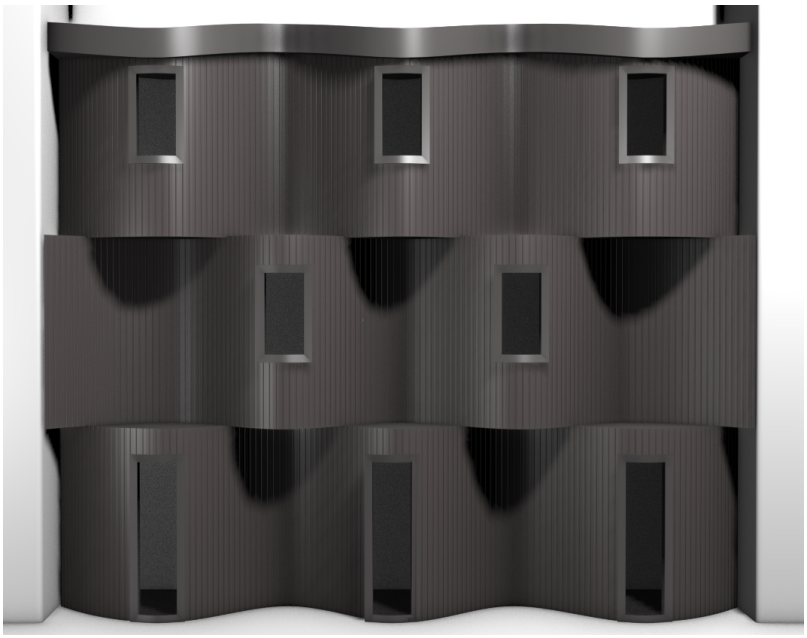
4. Wire the new spline node to the generator's **Clipping Area** input.



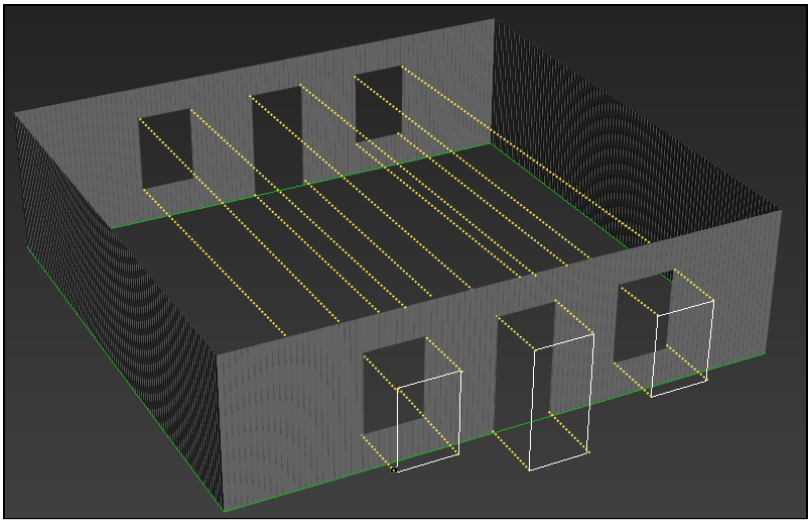
5. Nothing will happen, this is because the clipping operation is set to project the spline on the RailClone object's Z axis. To change this, select the **Generator** and go to the **Properties** panel. In the **Clipping Area** group, change the **Projection** axis to **Y**.



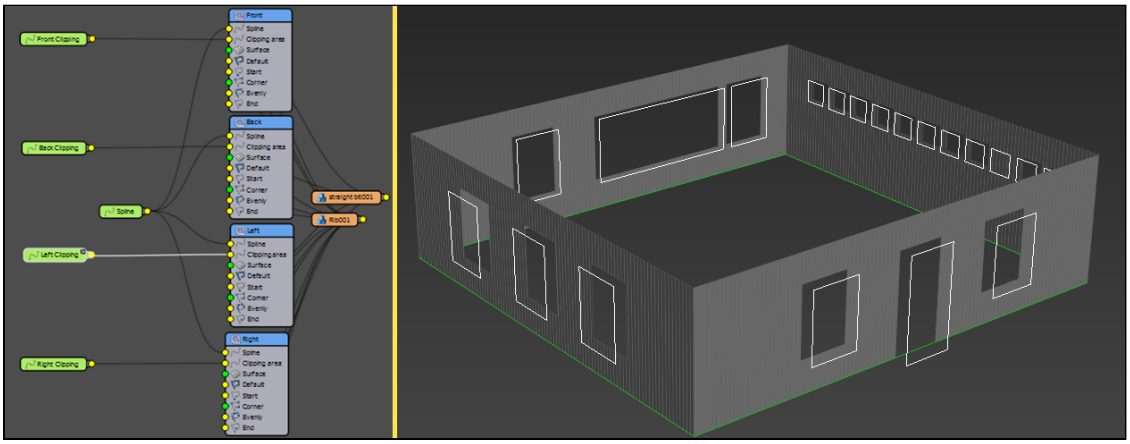
6. The clipping spline is now correctly cutting through the cladding.



You can choose from any of the X,Y or Z axis to best match the orientation of your RailClone object, however you can only project on a single axis per generator and the clipping operation will go through the model, removing geometry from both sides.



If you have an object that needs to be clipped from multiple angles you will need to split the style into different generators or duplicate the RailClone object. For example. a four sided building might use a separate generator for each façade, each with its own clipping spline. This will allow you to have different apertures on each aspect of the building.



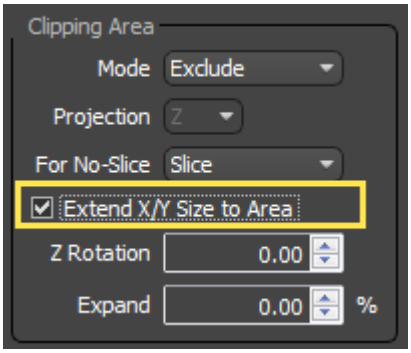
6.3 Using clipping splines to define the X/Y Area

Clipping splines can be used to automatically set the dimensions of a two dimensional array. This is the fastest way to create styles where you need to fill a spline area. Using this mode you do not need to use X/Y splines or enter a length and width value manually, making it ideal when using RailClone as a floor, ceiling or vertical cladding generator. In the next exercise we'll adapt the decking style from the beginning of this chapter to extract its size from a perimeter spline.

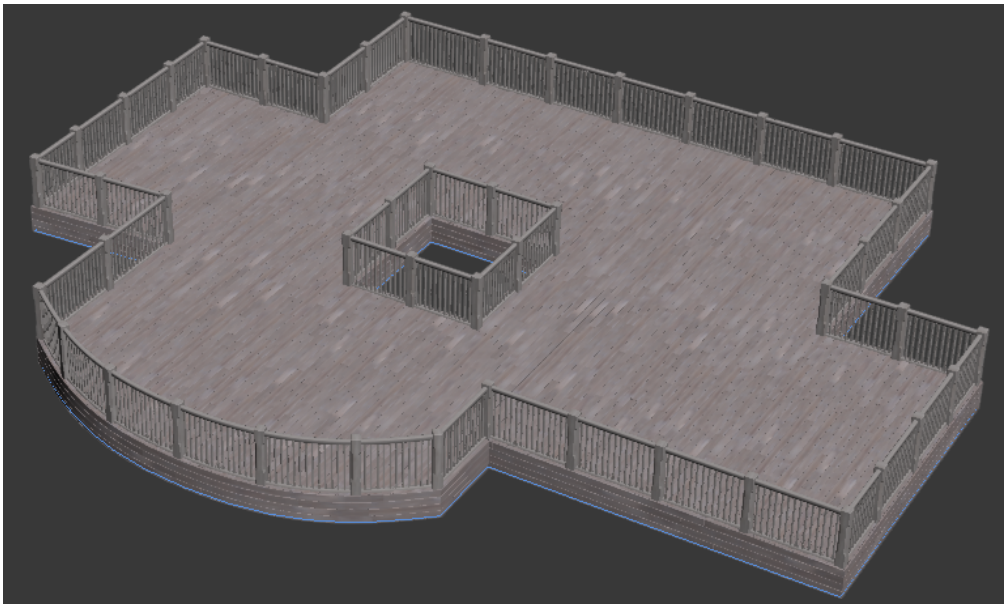
Exercise: Use X/Y Size Mode to create decking

1. Continue from the **chapter_6_deck_start** file used in the exercise above.
2. Select the RailClone object and open the Style Editor. Delete the existing spline nodes attached to the X Spline and Y Spline inputs of the Decking generator - we won't be needing them any more. At this point the geometry for the deck will disappear as there is nothing to determine the size of the array.

3. Select the Decking generator and go to the properties panel. Turn on **Extend X/Y Size to Area**. The deck will now re-appear inside the existing clipping spline.



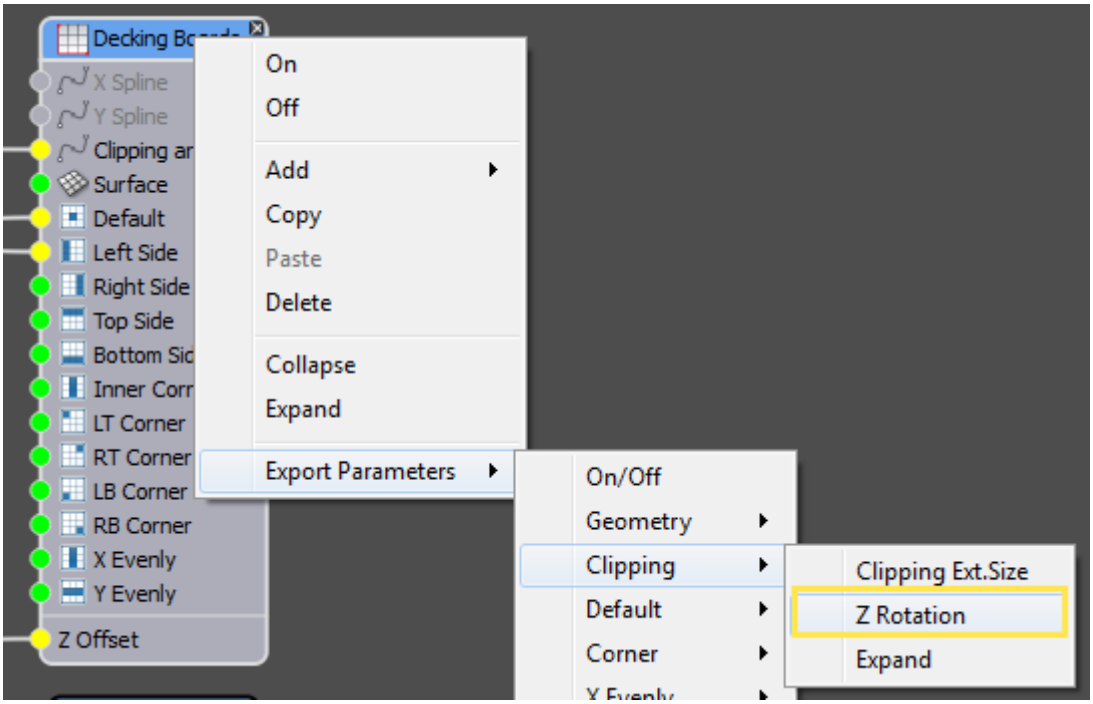
4. Let's test the style on a different spline. Close the Style Editor, then from the **Modify** panel go to the **Base Objects** rollout and select **Clipping Spline**. Click on the spline picker and select *deck_plan_spline* from the scene. The decking will update, automatically creating an array of the correct size and clipping the geometry to match the plan.



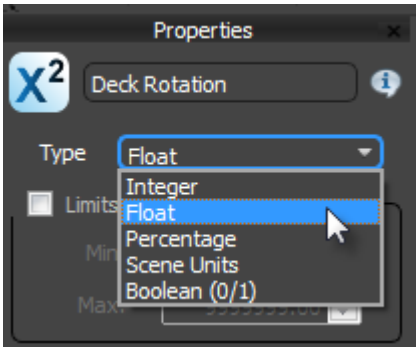
Exercise: using the Rotate XY Rotate

When **Extend X/Y Size to Area** a new Z Rotate parameter becomes available. This allows you to rotate the entire array inside the spline. As the generator rotates, the size is automatically updated to ensure that the entire spline is covered by the array and the geometry is clipped to fit with the new orientation. In this exercise we'll add an numeric node to this property so that it can be easily edited from the Modify panel.

1. Continuing from the previous exercise, right click on the Decking generator and **Export Clipping** > **Z Rotation**



2. Wire a new **Numeric** node to the exposed **Z Rotation** input. Change the type to **Float** and rename is *Deck Rotation*.

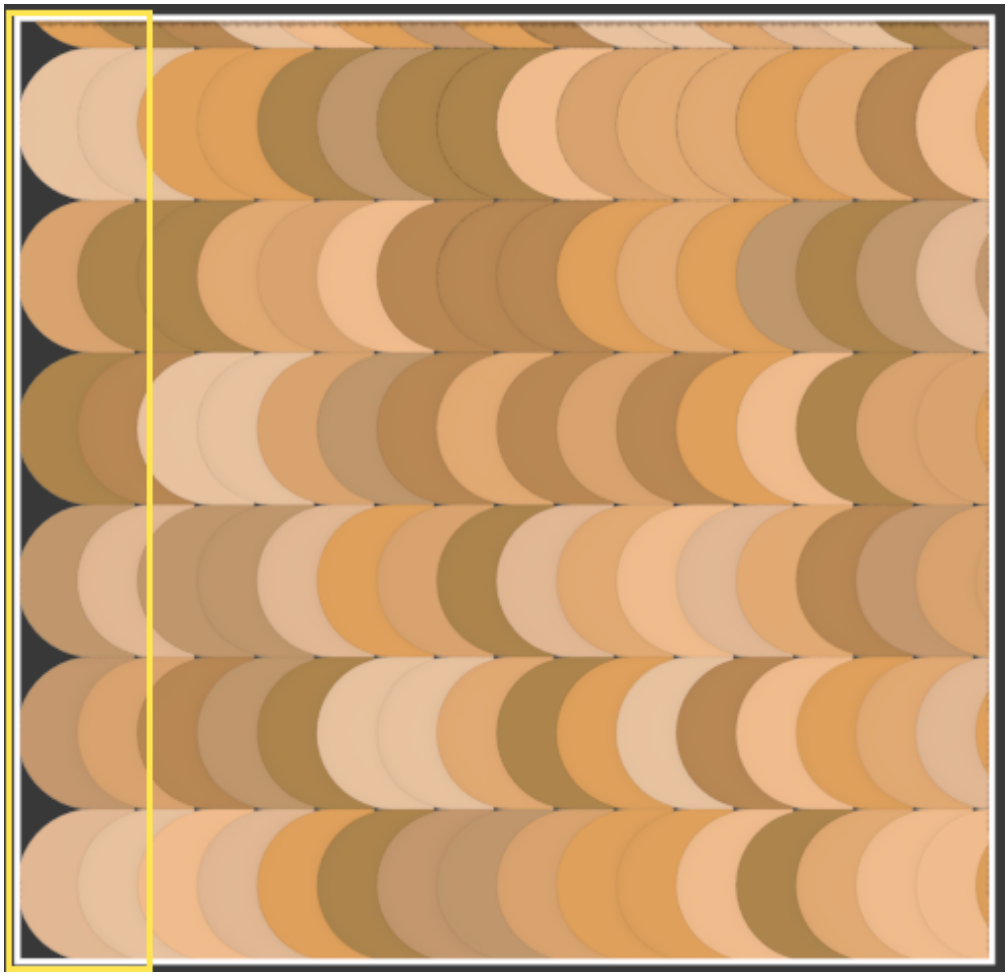


3. Now close the Style Editor. You can now edit the angle of the boards by going to the **Parameters** rollout and entering a new value for the **Deck Rotation** property.

6.4 Expanding areas - due to padding values or that shapes of certain segment's

A second parameter called Expand also becomes available when using **Extend X/Y Size to Area** . This allows you to adjust the automatically generated size of the array and is useful for situations where segments are missing or gaps appear around the perimeter of the spline. This

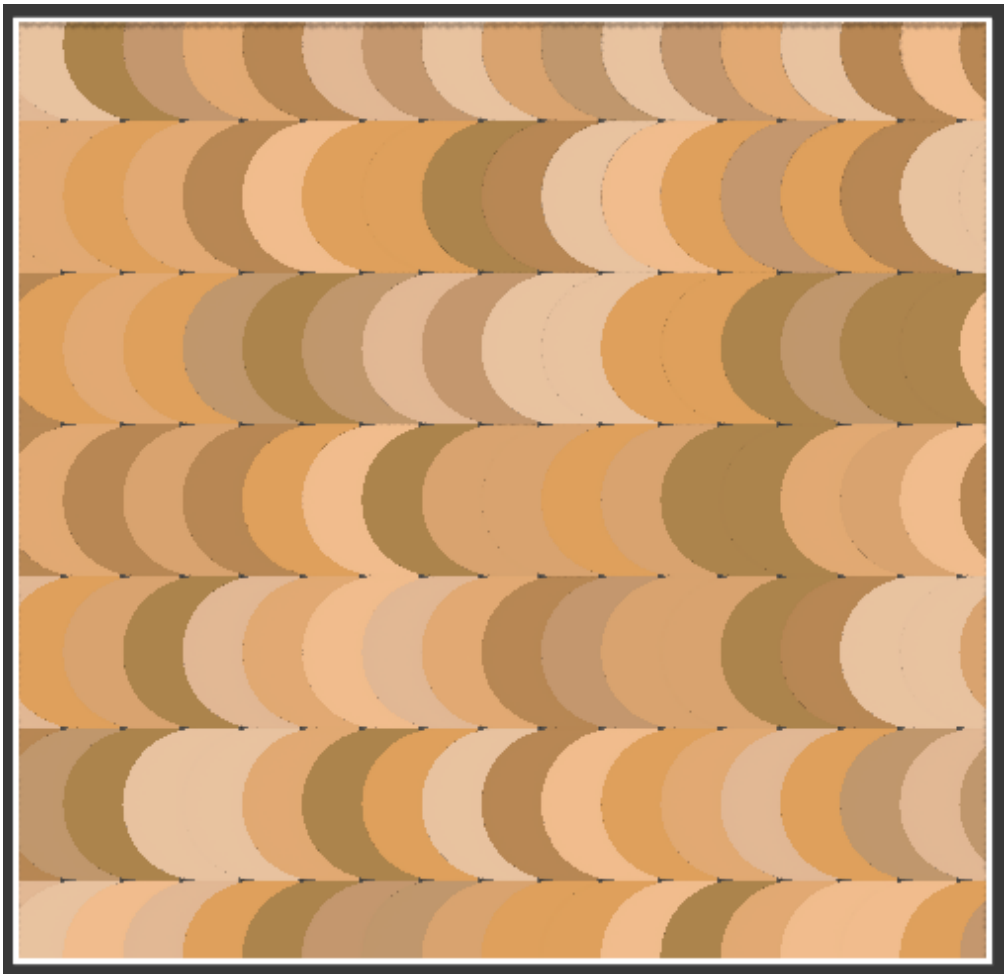
can happen when using padding and transform to adjust the positioning of segments, or if the side of the geometry being used is not parallel with the spline. For example, the tiling style shown in the diagram below features concave and convex areas that means it does not slice well on the left side of the array.



To fix this:

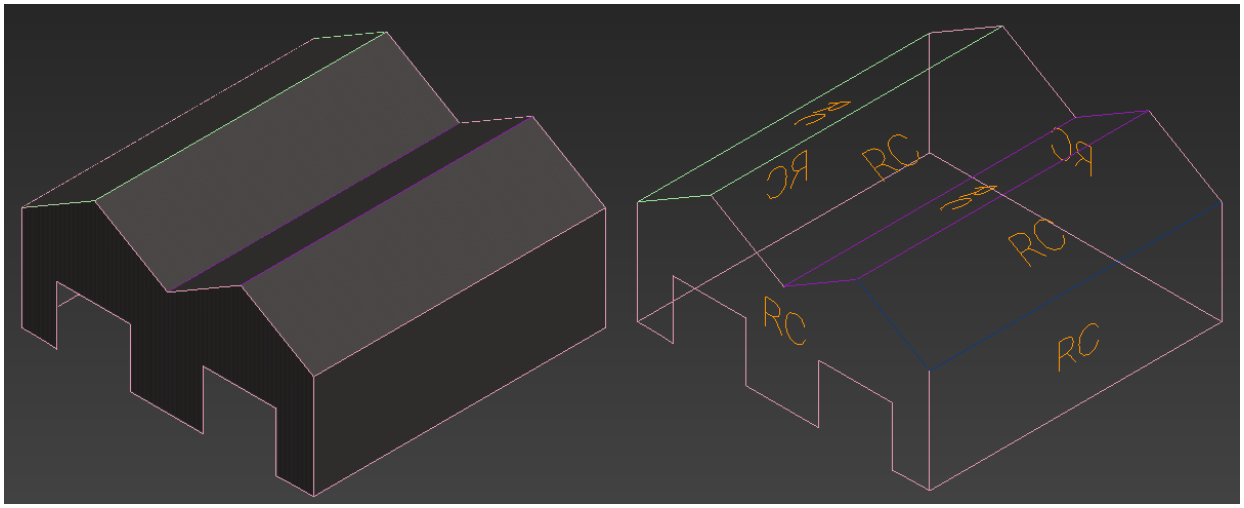
1. Open the file called **chapter_6_clipping_extend_start** from the downloads for this guide.
2. Select the tile RailClone object and open the **Style Editor**.
3. Select the **Generator**.

4. From the **Properties** panel, change the **Clipping Area** > **Expand** value to **15%**. As the array enlarges the gaps are gradually pushed to the edges of the clipping spline and disappear.

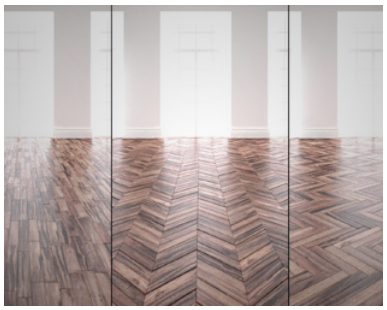


6.5 Using Extend X/Y Size to Area to create Vertical styles.

The most important thing to understand when working with clipping splines is that the projection axis relates to RailClone's **local** coordinate system. The axes of the spline itself are ignored. This means that if you want to create a vertical or angled style with RailClone using Extend X/Y Size to Area you must rotate the RailClone object so that it is oriented correctly. To see an example of this, open *chapter_6_cladding_vertical*. This scene uses multiple RailClone objects to add a simple cladding style to the sides and roof of a structure. In the scene there is also a version where the splines have not yet been selected so that you can see how the RailClone objects are aligned in relation to their clipping spline in order for them to work correctly.



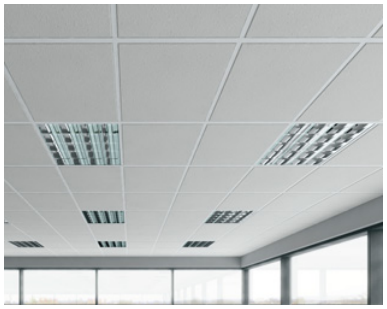
6.6 Related Tutorials



create_a_parquet_floor

Create Parquet Floors

Follow this tutorial to create three types of parquet floor. These styles all use a perimeter spline to automatically set the size of the array.



creating_ceilings

Creating Ceilings with RailClone

Follow this tutorial to create a suspended ceiling that uses a perimeter spline to automatically set the size of the array.

Documentation

For more information please see the clipping area section of the 1D arrays - Generator L1S and 2D arrays - Generator A2S sections of the online documentation.

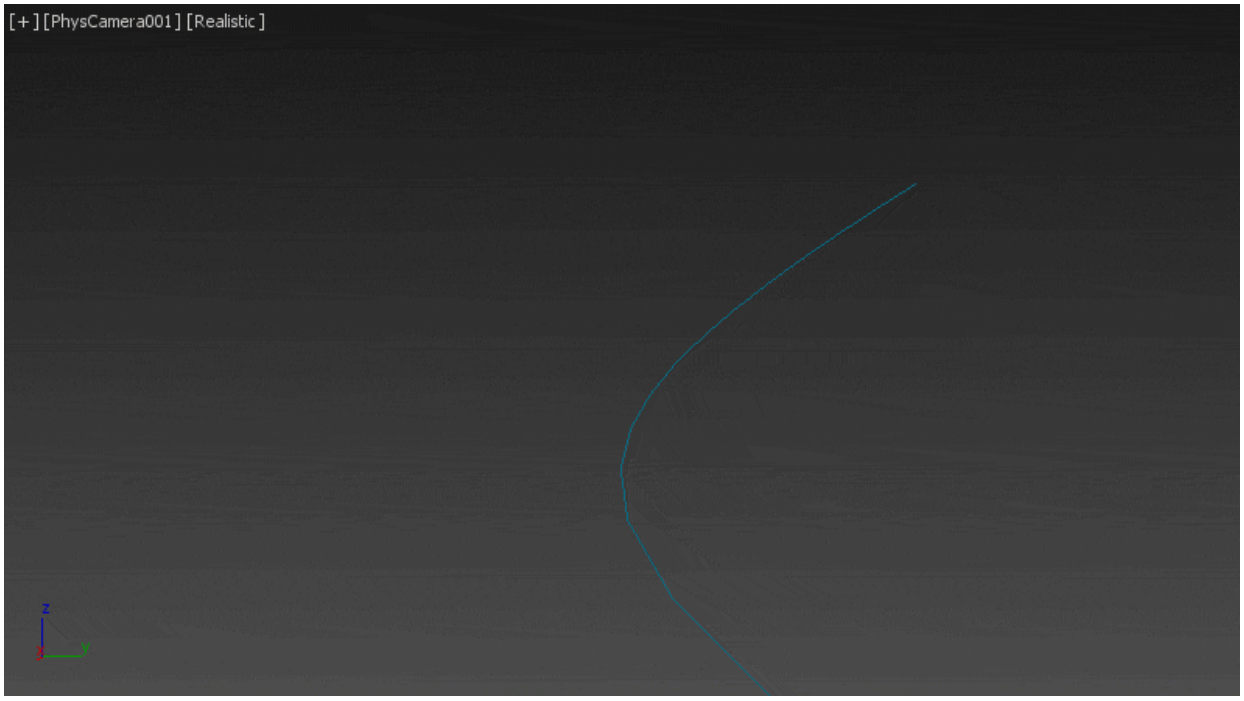
7 7 - Master Spacing Modes

Many parts of arrays are placed precisely according to the base spline and size. For example Start segments are placed on the first vertex of a base spline, End Segments appear on the last vertex and corners can be placed on intermediary vertices of a particular type. There are however two types of segment that not placed with such precision, these are used to fill in between these set markers. Evenly segments are placed automatically at regular intervals on the X or Y axis of the array, and Default segments fill in any remaining spaces. Though distribution of these segments is automatic, RailClone has several useful rules to help you control the placement and scale.

In this chapter we'll take a look at these parameters using a sci-fi corridor example. You can find this scene in the downloads for this guide, the files is called *chapter_7_spacing_modes_start.max*.

This scene contains 7 RailClone objects. Each object is relatively simple in itself, but contributes an extra element to the corridor style which builds up to create a more complicated looking final result.

In the finished version, *chapter_7_spacing_modes_end.max* file, all the RailClone objects are combined into a single style and the corridor can controlled from the same interface and applied to a spline with single click. In the start file each generator has been separated to individual RailClone objects to maintain compatibility with RailClone Lite.



In the start file all the generators use the default Tiling mode. Let's change these parameters to explore how the settings work.

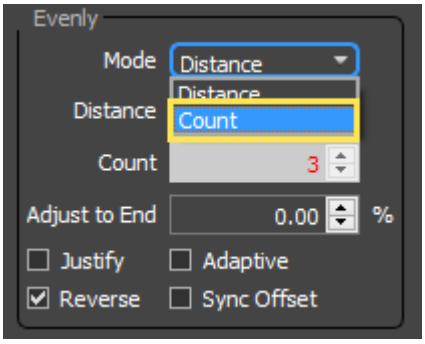
7.1 Exercise: Exploring Evenly Spacing.

Evenly segments are distributed at regularly spaced intervals along the X and Y axis. To control the distance between evenly segments you have two options, **Count** or **Distance** mode.

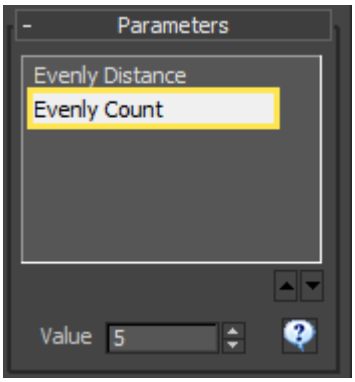
Count mode enables you to divide the spline path into a fixed number of evenly spaced segments. Let's try this out in the scene file.

1. Open *chapter_7_spacing_modes_start.max* from the downloads for this tutorial.
2. Select the RailClone object called *rc_walls* (or rc_walls_lite if you don't have the Pro version)
3. Go to the **Modify** Panel, and apply the style to the *flat_spline* path from the Base Objects rollout

4. Open the Style Editor and select the Generator. Go to the Rules tab in the **Properties** panel and you'll find the **Evenly** settings at the bottom of the interface. Change the mode to **Count**.



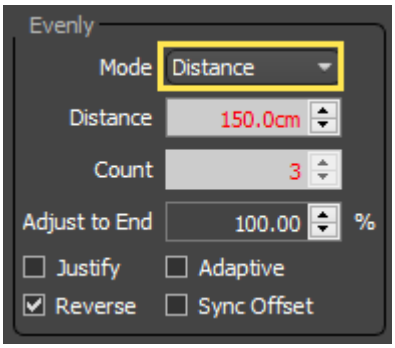
5. Count mode divides the length of the path equally using the Count value. To change this you can edit it directly in the Style Editor, or you can export and attach a Numeric node so it can be updated from the Modify panel. We've already exported this property, so go to the Paramters rollout in the Modify panel and enter a value of **5** for **Evenly Count**.



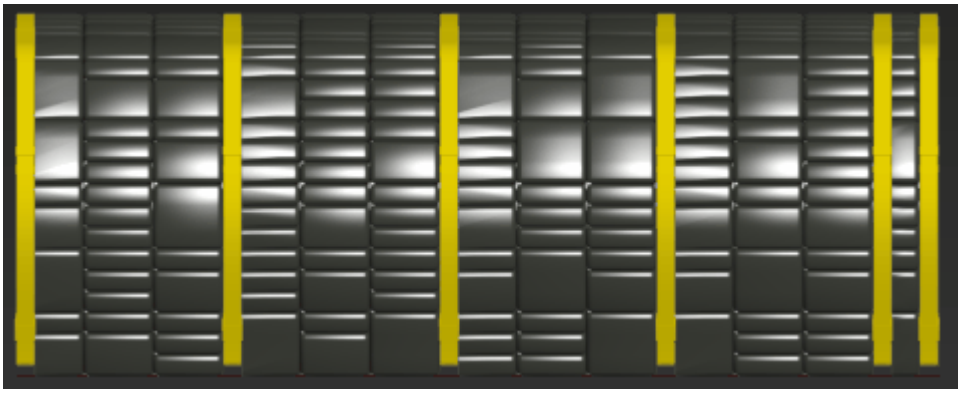
6. The path's length has been animated and if you scrub the timeline you'll see that irrespective of the length of the path, the number of evenly segment's remains exactly the same.

Alternatively you can use **Distance** mode. When this is active the number of evenly segments is not fixed, instead segments are added at regularly spaced intervals along the spline separated by a measurement set by the distance value. As the length of the spline increases or decreases evenly segments are added or subtracted accordingly. To change the corridor walls to use distance mode, follow these steps:

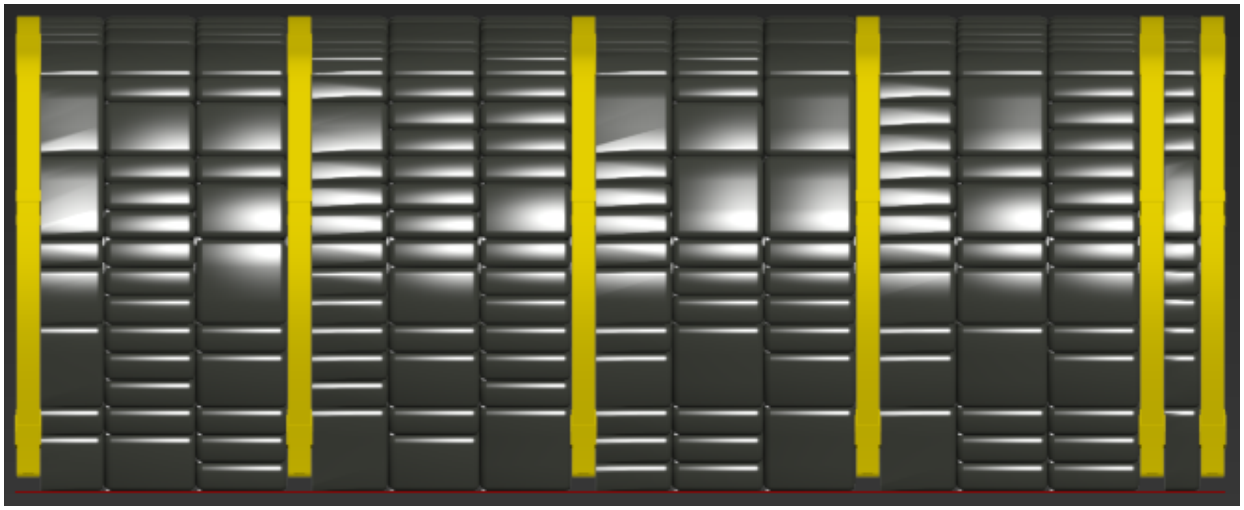
1. In the Style Editor, change the **Rules** > **Evenly** mode to **Distance.**



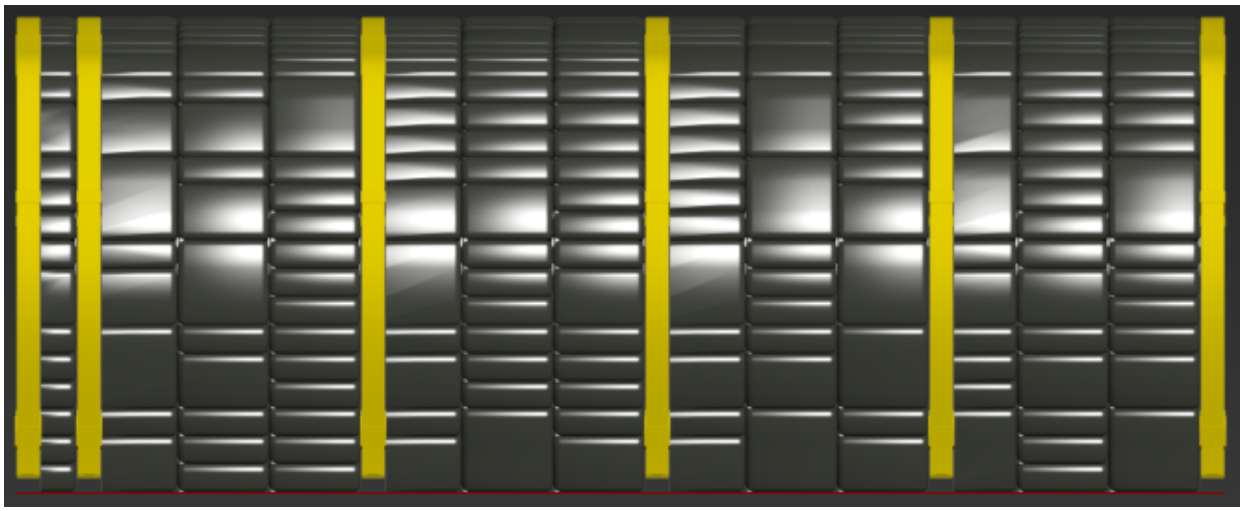
2. Change the **Distance** value to **150cm**, you can do this directly in the style editor or export the value and use a numeric node to make it editable from the Modify panel.
3. Now scrub the timeline. As the spline size gets larger you'll see additional evenly segments added to the array. Try different distance values to change the spacing between Evenly segments.



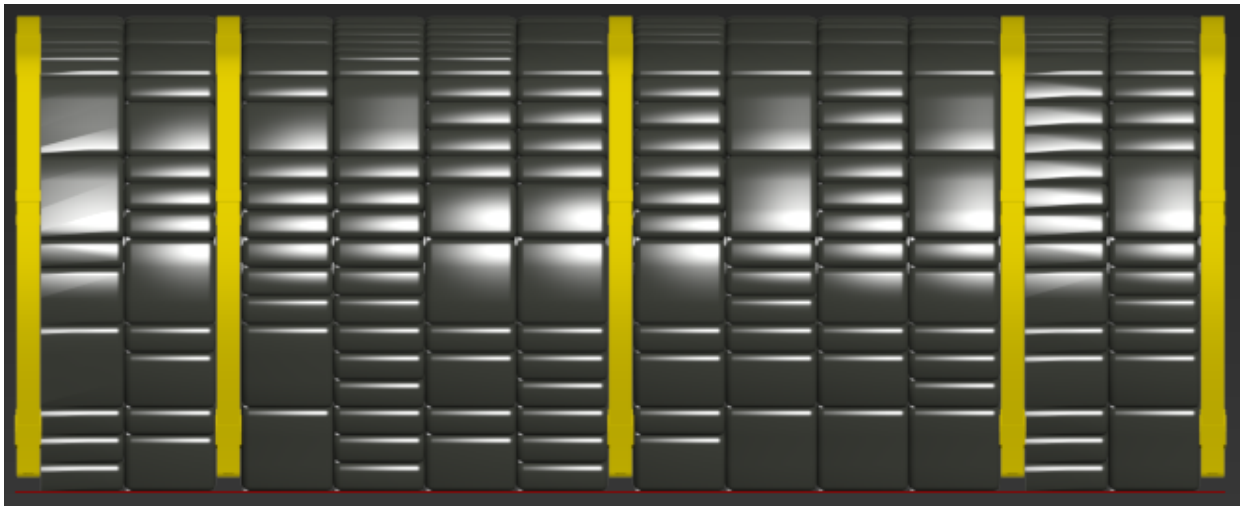
In distance mode you have 4 additional settings that control how evenly spaced segments are distributed along the spline. At its most most basic, evenly segments are equally spaced at regular intervals starting from the beginning of the spline. This can result in an asymmetric distribution if the spline's length doesn't divide equally into the Distance value, as you can see in the example below.



This is not always undesirable and it is possible to change the direction of the distance calculation by selecting the **Reverse** checkbox. In the image below you can see the effects of this, notice now that the final evenly segment is now on the left.



It is also possible to automatically create a symmetrical distribution of evenly segments using **Justify** mode. When this is active the spacing of the first and last segment is identical as shown below. This saves you time adjusting evenly distances and can be used to create symmetrical designs for styles that use multiple paths.

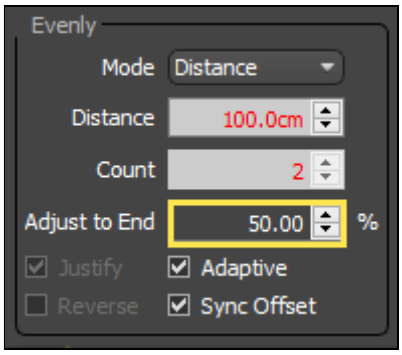


Finally adaptive mode can be used to ensure the spacing between Evenly segments is identical along the whole spline. This mode calculates the number of times the full Evenly distance can be distributed along the path then divides that number by the path's length to generate a new equalised spacing. This is the mode we'll use for all the generators in this exercise, to set it:

1. Select a Generator, go to the **Properties** panel and select the **Rules** tab.
2. Change the **Evenly Mode** to **Distance.**
3. Turn on **Adaptive**. All the bays will now be equally spaced.

How does RailClone know when to add a new evenly segment?

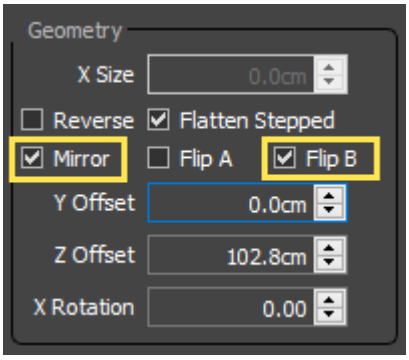
This is handled by the Adapt to End setting that determines the amount of space that can remain at the end of a path before a new evenly segment is added. This is measured as a percentage of the distance value.



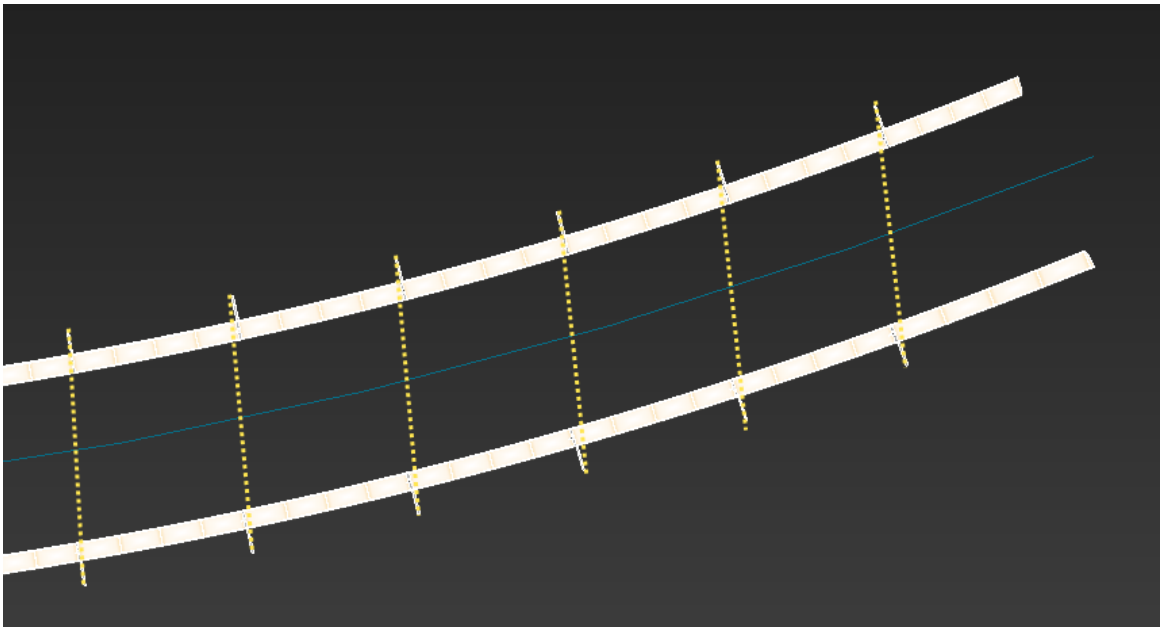
7.2 Exercise: Sync Offset

If you use the Generator's Y Offset features then RaiClone uses the length of the offset path to calculate the position of evenly segments. If you have a mirrored and offset object on a curved path you are therefore creating two new paths, one of which is shorter than the other. This can result in segments that are not properly aligned on both sides.To demonstrate how to fix this, we'll take a look at the lighting style in the scene files for this chapter that use this technique.

1. Select the RailClone object called *rc_lights*.
2. From the **Base Objects** rollout, assign the style to the path named *curved_spline*.
3. Open the **Style Editor** and select the generator.
4. From the Properties panel, turn on **Mirror** and check **Flip B**

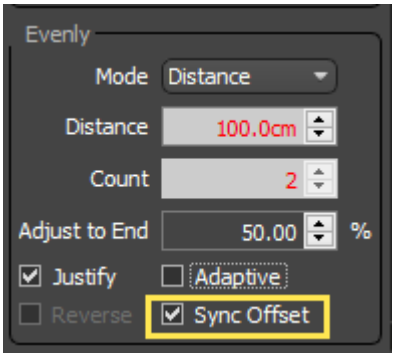


5. Enter a value of 42.6 for the **Y Offset** parameter. You will now two rows of lights, one either side of the central spline.

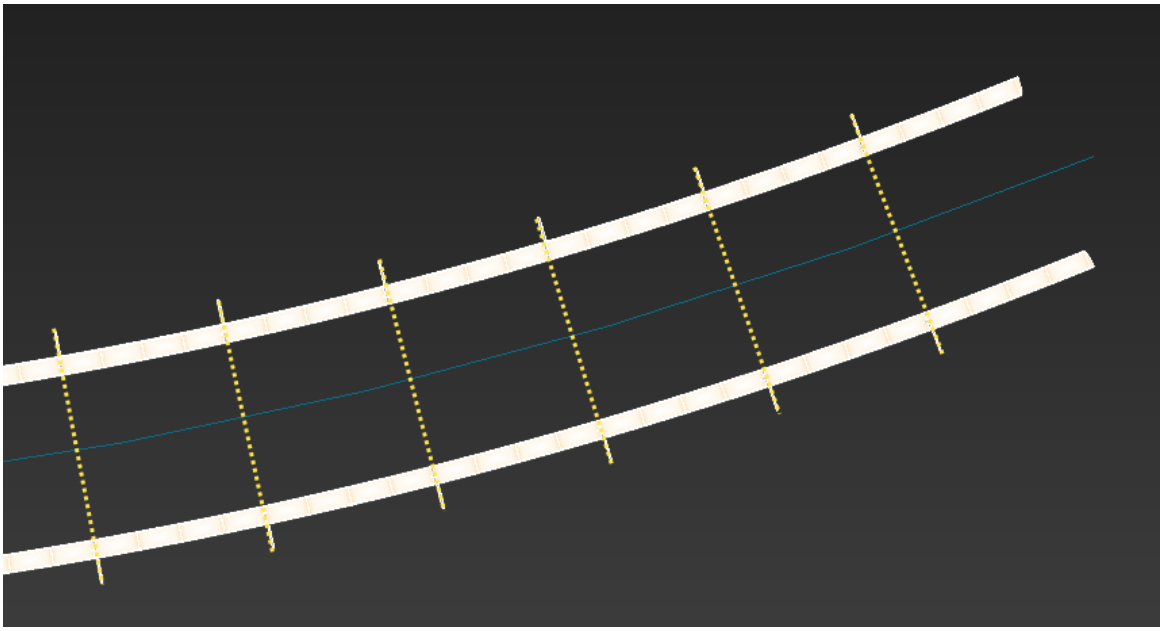


However there is a problem, the connection brackets are not aligned correctly as each row is calculated independently. The connectors should be aligned perpendicular across the spline.

6. To fix this, go to the Rules tab and turn on **Sync Offset**



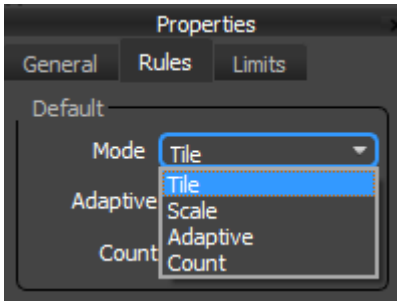
The evenly positions for both rows are now calculated based on the central spline and the Segments are translated into the correct position perpendicular to the path.



7. To finish up this section, change the path for the lights and rc_walls to curved_spline. The walls and lights are now complete.

7.3 Exercise: Exploring Default Segment Tiling

The remaining area, i n between the Start, End, Corner and Evenly parts of the array, is filled with Default segments. Exactly how these are distributed is defined by the settings found in the Default Mode group.

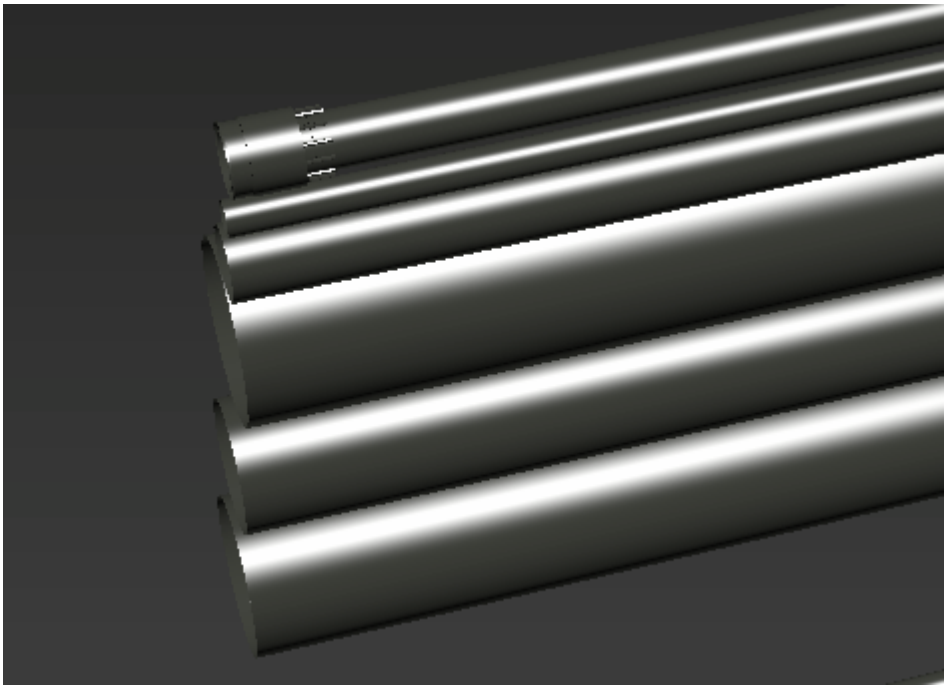


There are 4 modes, each of which determines how the object is repeated and scaled to fill the available space. In this exercise we'll use this settings on the remaining parts of the corridor to illustrate how these settings can be used.

Tile (Default)

In the tile mode, default segments are repeated along the path. When a segment is interrupted by an Evenly, Corner or End part of the array, is is simply sliced to ensure a perfect fit. In our corridor style we will use this technique on the vertical pipes attached to the walls.

1. Select *rc_wall_pipes* and apply it to the *straight_spline* path.
2. Open the Style Editor and ensure the **Default** mode is set to **Tile**.
3. Scrub the timeline, as the spline changes length and the segments pass the end of the path you will see they are sliced.

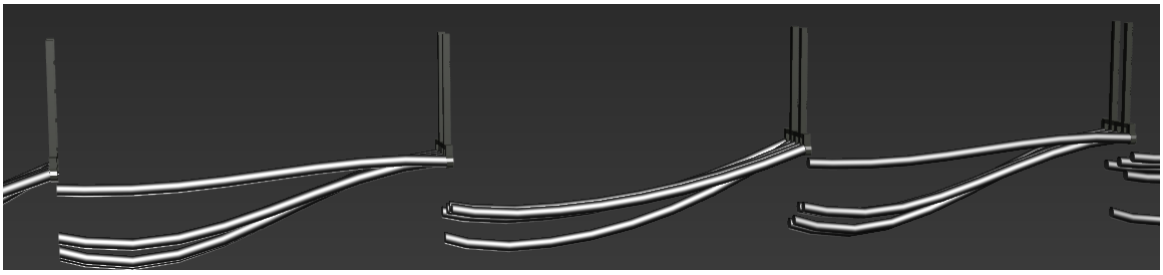


4. Change the path to *curved_spline* to add this style to the final corridor.

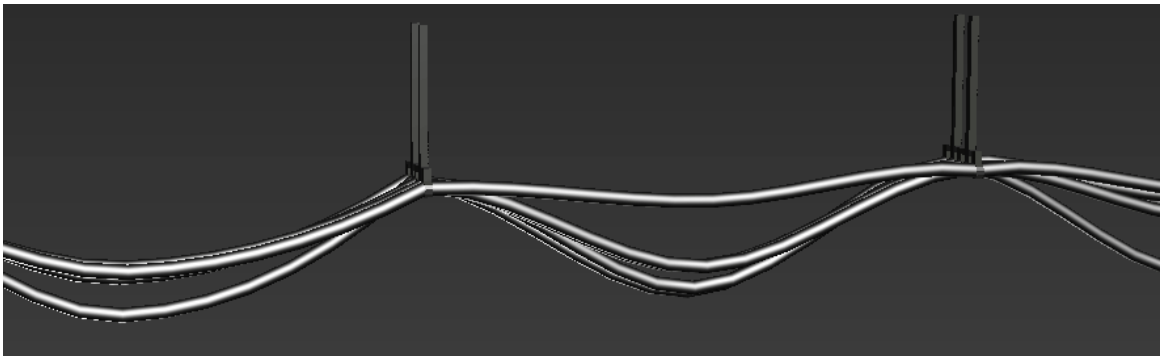
Scale

Instead of tiling multiple segments you can use **Scale** mode to add a single segment and then scale it to fill the space between the initial and final positions of the path section. Remember, a section is defined as the distance between Start, End, Evenly or Corner parts of the array and in this exercise we'll use scale mode to add the wires and pipes between Evenly spaced support brackets.

1. Select *rc_wires* and apply it to the *flat_spline* path. You'll see that in the Default tile mode the cables are sliced incorrectly



2. Open the style editor and change the **Default** mode to **Scale.** A single cable segment is now stretched between the supports and the style is correct. Even if you change the evenly Distance, the cables re-scale to match.



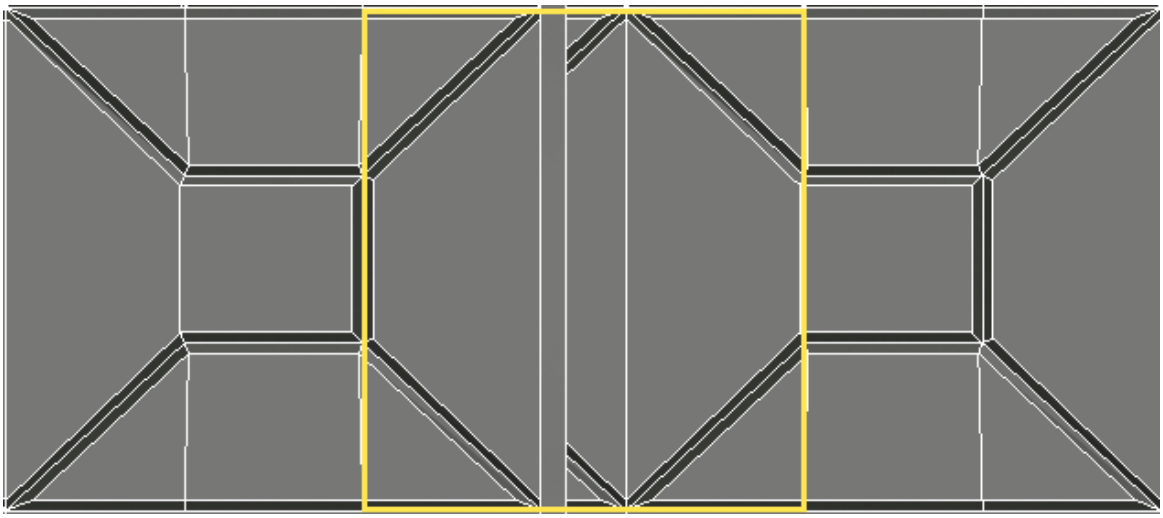
3. Change the path to *curved_spline* to add this style to the final corridor.
4. Repeat this process for rc_ceiling_pipes.

Adaptive.

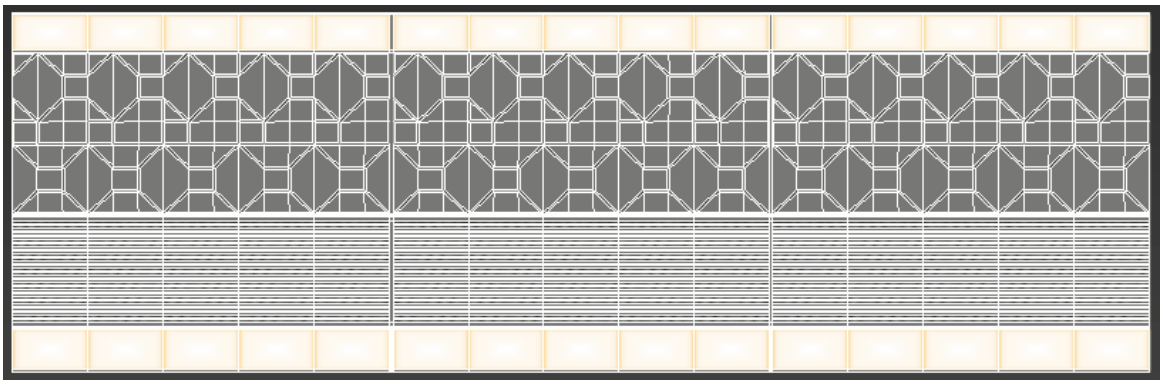
Adaptive mode is a hybrid between Scale and Tile. It calculates the number of whole segments that would fit into a path section and then scales them all equally on the X axis to fill the available space. This mode generates complete segments, with no clipping at the start or end of a spline segment. We can use adaptive mode on our floor tiles to makes sure that the pattern is not disrupted by incorrectly sliced geometry.

> When Adaptive mode is active, Corner > Bevel settings are unavailable. In the next section we will look at a workaround for this limitation.

1. Select rc_floor and apply it to the *flat_spline* path. You'll see that in the default tile mode the floor tiles are sliced incorrectly and the patterns do not quite line up.



2. Open the style editor and change the **Default** mode to **Adaptive.** Only whole tiles are now used and they are equally scaled to fill the available space. Even if you change the evenly Distance or spline length, new segments are added and scaled to ensure a perfect fit.



3. Change the path to *curved_spline* to add this style to the final corridor.

Adaptive Percentage

In Adaptive mode the Adaptive Percentage value specifies the minimum length of remaining space (expressed as a percentage of the segment's original size on the X axis), at which a new segment will be added. For example if a segment's X-dimension is 1m and the path section length is 4.25m, the segment will fit into the distance 4 times (4 x 1m = 4m). The remaining space is 0.25m, therefore a new segment will be added if the Adaptive Percentage is set at or below 25%.

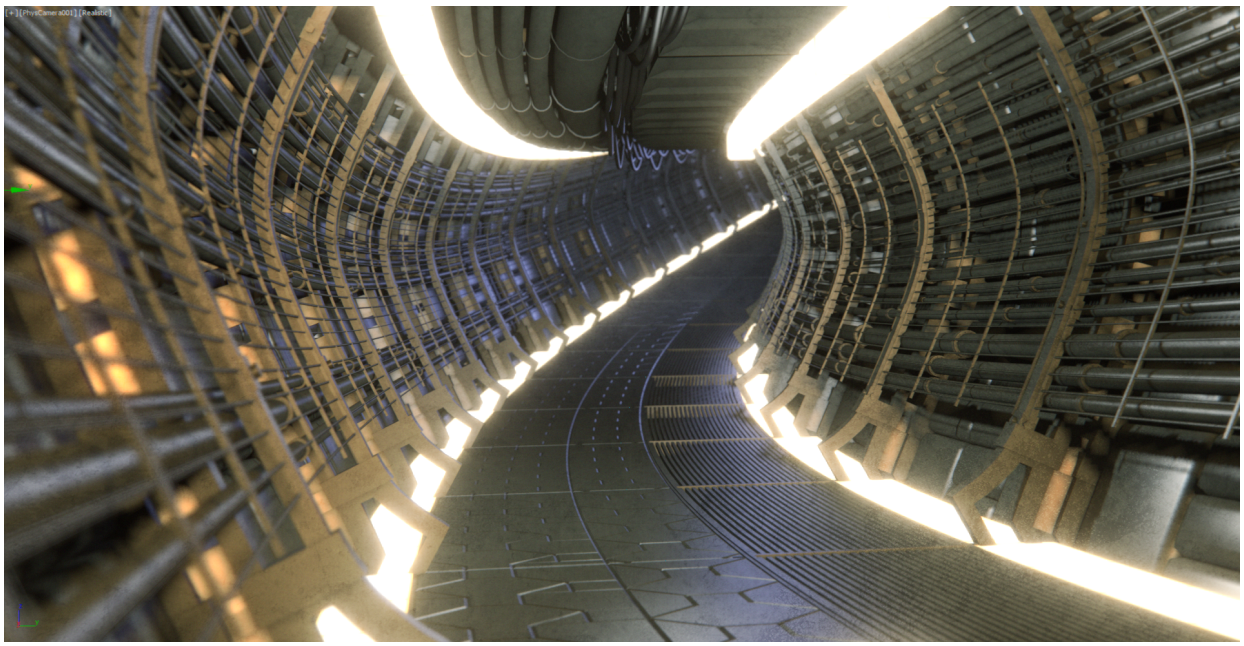
Count Mode.

Count mode scales a fixed number of segments to fit between the start and end positions of a path's section. In this example we will use this mode to ensure there are always 3 ducting boxes between the supports.

1. Select rc_ducting and apply it to the *flat_spline* path.
2. Open the Style Editor and change the **Default** mode to **Count.** You can now change the number of segments to scale on a path section using the Count value directly inside the Style Editor, or by exporting the value and connecting it to a numeric node so that it can be easily accessed from the modify panel. Set this value to 3 and you will always get 3 ducting boxes between the supports even if you change the length of the spline or evenly distance.

3. Finally, to complete this exercise change the path to *curved_spline* to add this style to the final corridor.

4. Press C on the keyboard to change to the camera view and see the inside of the finished sci-fi corridor.

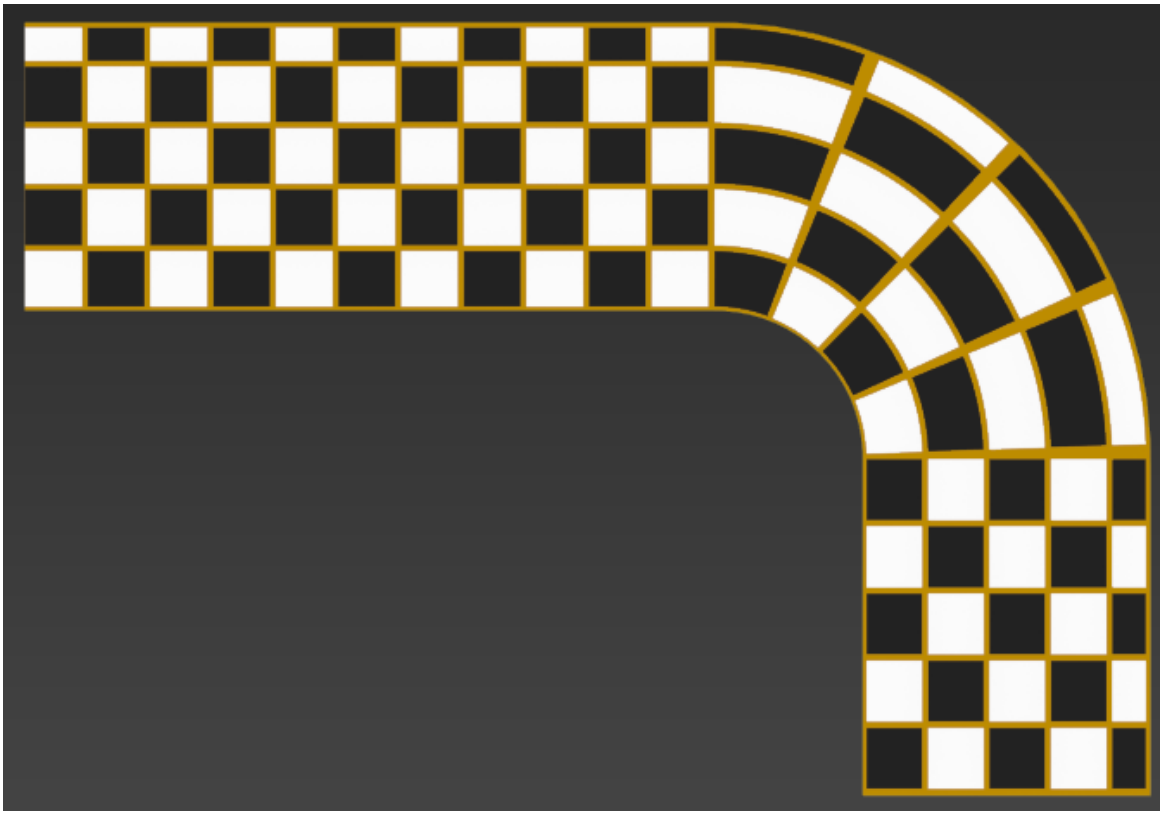


Because Adaptive, scale and count mode prevent RailClone from slicing geometry it has the potential to improve the number of segments that remain instanced. Any sliced segment contributes to the overall polygon count whereas scaled segments do not. Of course this does not apply if the segment have also been deformed to follow the spline in which case they will also contribute to the overall polycount.

7.4 Y Mode

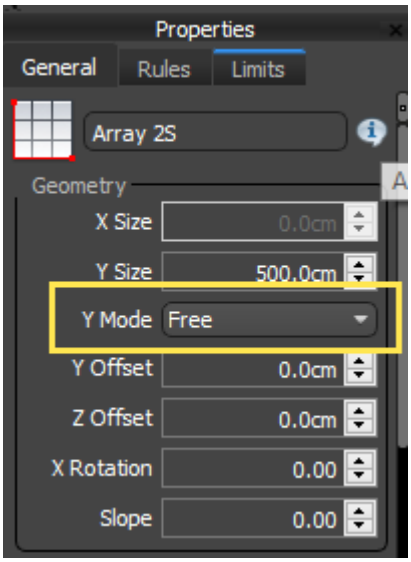
To round up this chapter we'll take a look at one last mode that can affect the distribution of Default segments. The A2S generator has a little known **Free Alignment Mode** that allows you to calculate the spacing of default segments for each row independently. To see how this is useful we'll look at typical paving style.

1. Open *chapter_7_free_mode.max* from the downloads for this guide.You'll find a simplified paving style with alternating black and white tiles to accentuate the behaviour of the default segments.

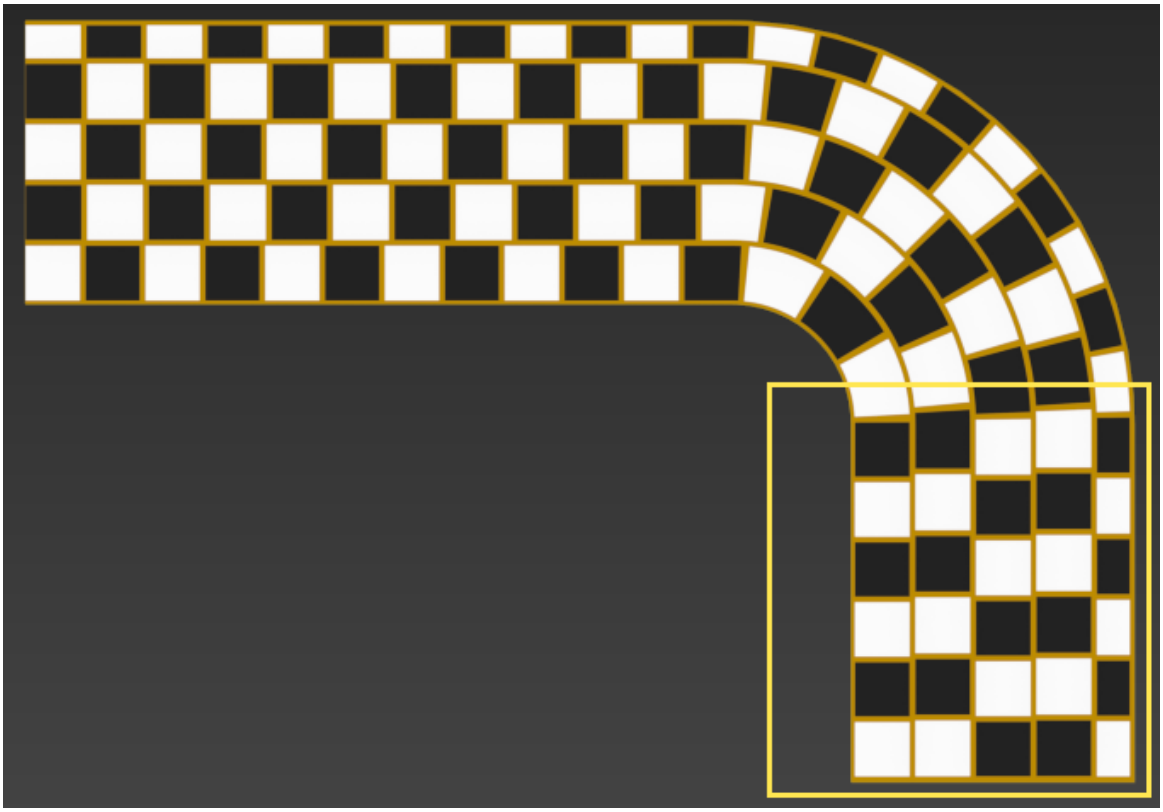


Notice that on the corners the tiles get larger the further out from the corner they are. This is because the style is using the default **Aligned** mode in which segments are scaled so that the same number of them appears in each row.

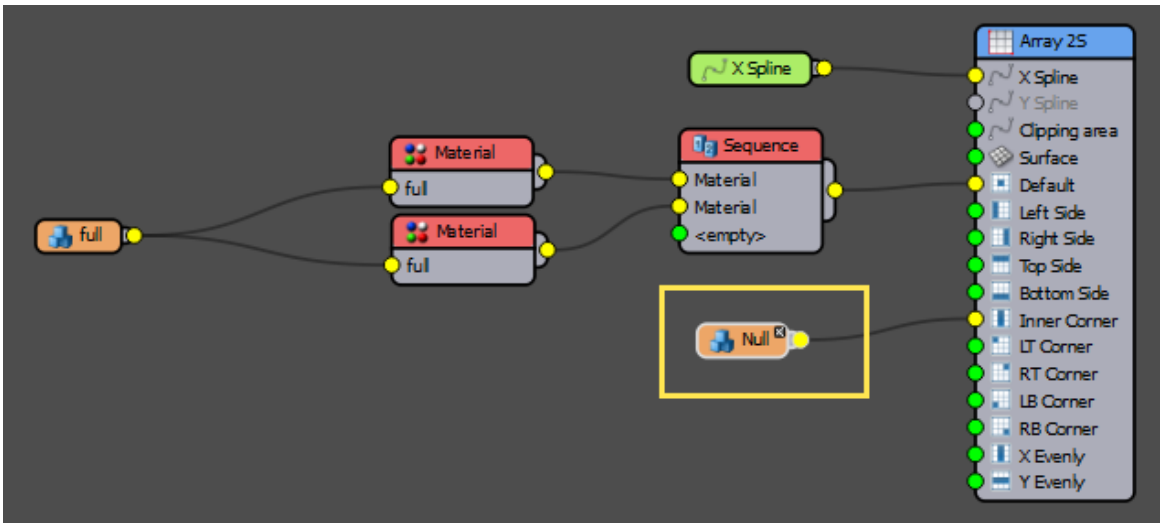
2. To fix this we can force each row to calculate its own default spacing. Just open the Style Editor, select the **Generator**, and change the **Y Mode** to **Free** from the properties panel.



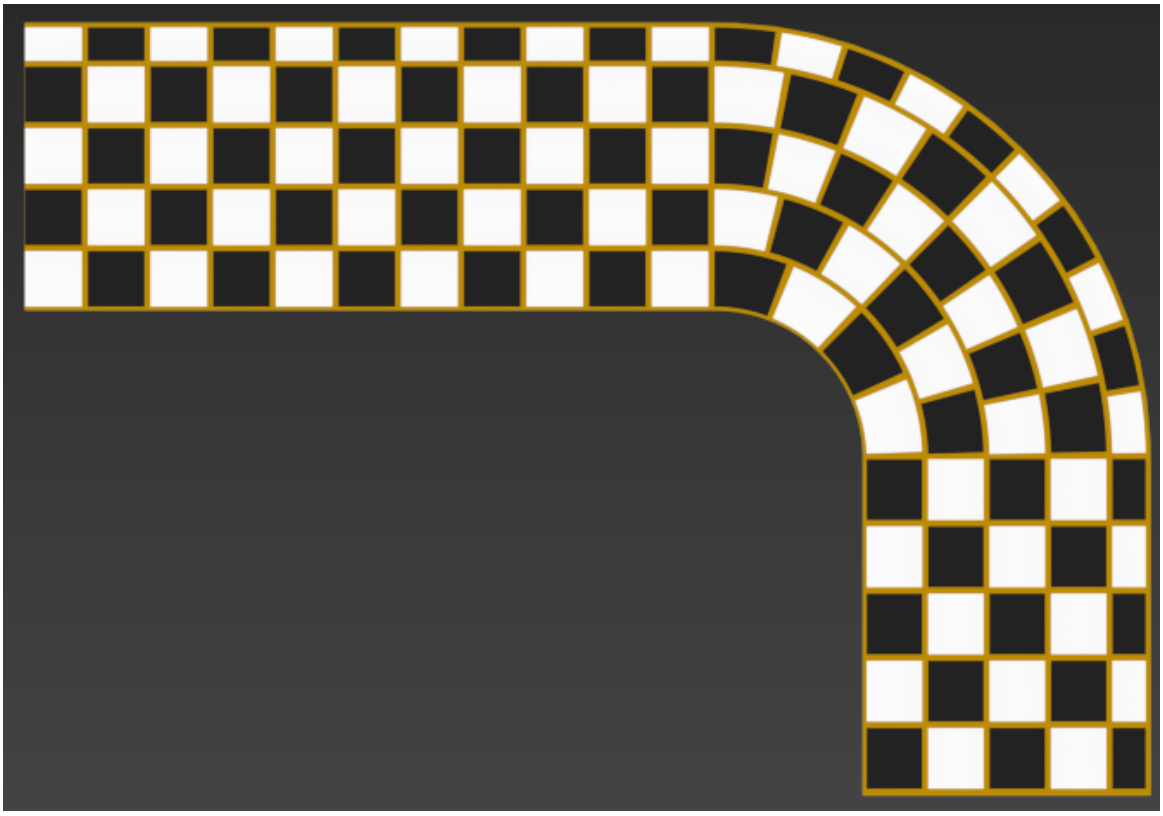
This fixes the corner but you can see that we've lost the checker pattern after the bend:



3. To return the pattern we need to break the array into different path sections, one before the bend, the bend itself, and then one afterwards. At the moment because there are no Evenly of Corner segments the entire spline is considered to be one section. Fortunately the solution is very simple. Just wire an empty Segment operator to the Corner input to break the path at that point.

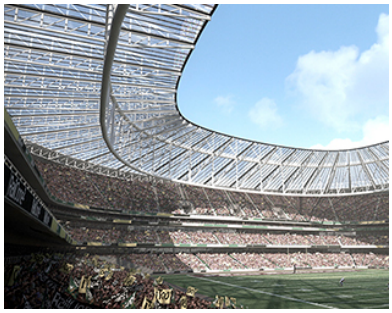


4. Each row in the corner is now calculated separately but the pattern is reset on the straight sections!



In this section we've looked at all the setting found in RailClone that influence the distribution of automatically repeating segments. Other segments are placed according to the start, end and vertices of the base spline or the size of the array. In the next chapter we'll take a deeper look at some of these as we explore how to master the controls for corners.

7.5 Related Tutorials



stadium

Stadium Tutorials

The five-part stadium tutorial is a great introduction RailClone that features many tips to help you get the most of the spacing modes.

power_lines

Power Lines Tutorial

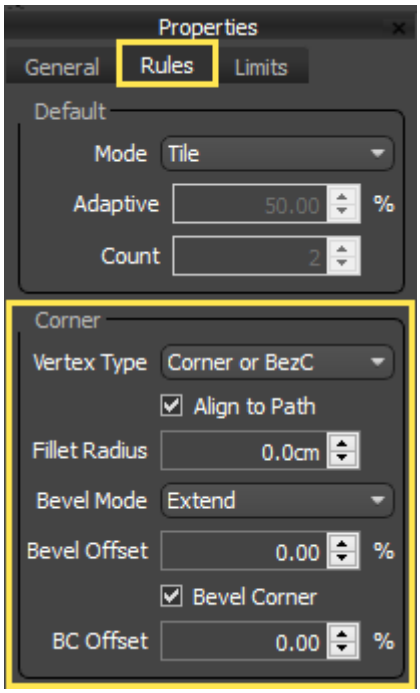
Learn how to use scaling mode to create power lines. This tutorial is similar to the cables style found in the sci-fi corridor

Documentation

For more information please see the 1D arrays - Generator L1S and 2D arrays - Generator A2S sections of the online documentation.

8 8 - How to Fine Tune Corners

The L1S generator's Corner input and the A2S generator's Inner-Corner input allow you to position segments based on the vertices of the path. Though this seems straightforward, there are a number of controls to help you tailor your corners exactly how you need them. All the settings to control the behaviour of corners can be found in the generator's properties by clicking on the **Rules** tab.



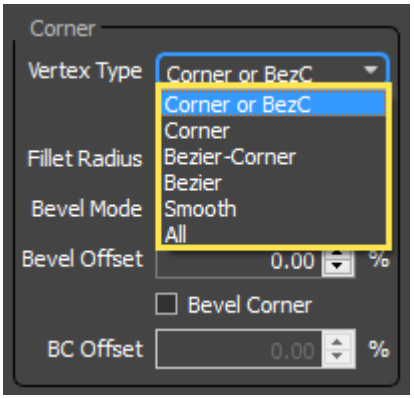
8.1 Choosing the vertex type.

First of all, RailClone lets you decide which types of vertices are going to generate corner geometry. 3ds Max's spline objects have 4 types of vertex: corner, bezier-corner, bezier and smooth. In the generator's properties you can choose which of these 4 types creates a corner segment by simply selecting them from a drop down list.

We'll look at an example to see how this works. In the following few exercise we'll create a style with a path, signposts, benches and bins to illustrate how RailClone handles corners.

Exercise: Changing vertex types

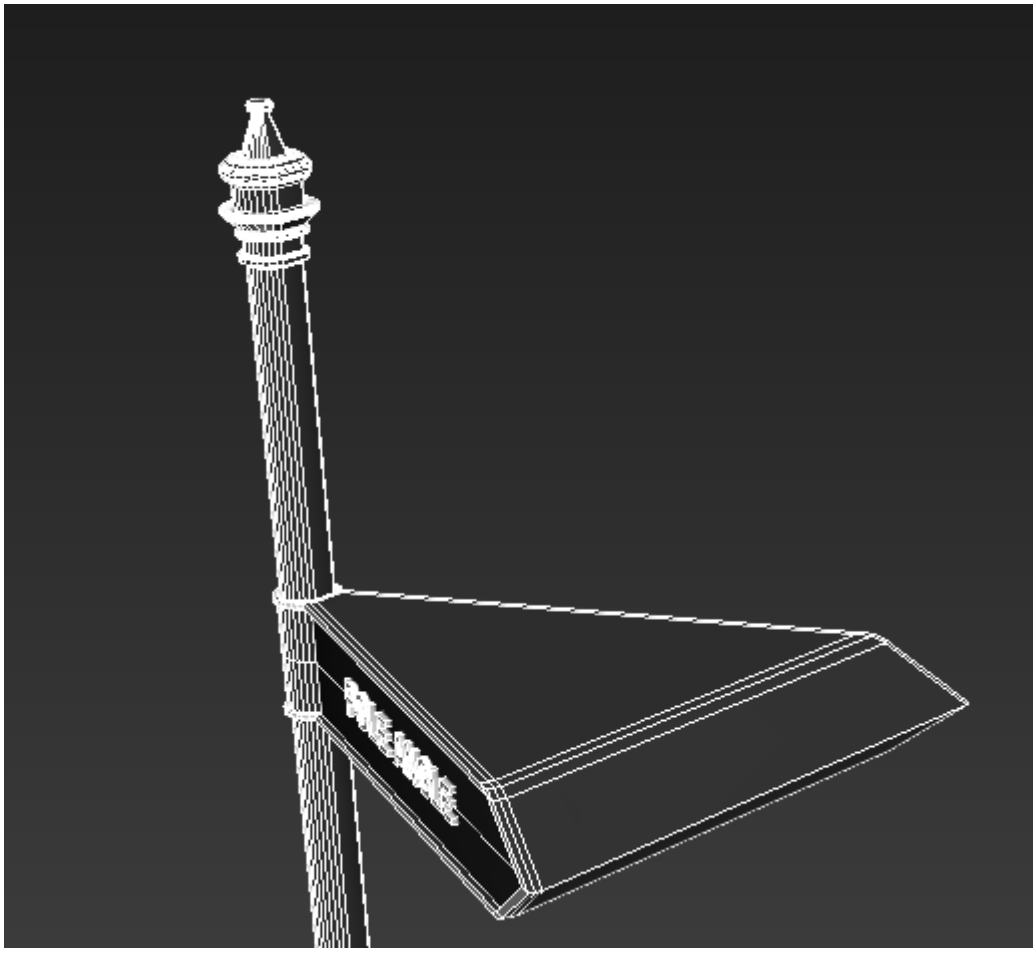
1. Open ***chapter_8_signpost_corner_angles_start.max*** from the downloads for this guide.
2. In this scene you will find two RailClone objects, a path and separate style for some sign posts. These could have been combined but to maintain compatibility with Lite users they have been separated for this tutorial.
3. Select the RailClone object called *rc_signs* and open the Style Editor. All the segments are already added to the graph, but nothing is wired up yet.
4. Wire the *Post* segment to the Generator's **Corner** input. This segment uses a simple pole with an attached sign. It has been oriented so the the sign is perpendicular to the path.
5. You will now have a pole on every vertex on the spline of the corner or bezier-corner type. This is the default mode but it can be easily changed from the **Vertex Type** dropdown found in the **Generator's Properties** > **Rules** menu.



6. To test these out try changing the **Vertex Type** to **Smooth**. You will now only have a single pole on the spline's sole smooth vertex. Change the type again to **Bezier-Corne**r and you'll see 3 poles, and change it back to **Corner or BezierC** and you get poles only on the hard bends, which is where we want them. Being able to select the vertex type in this way allows you to place Corner segments where you need them, but still enable you to use other vertex types to control the shape of the path.

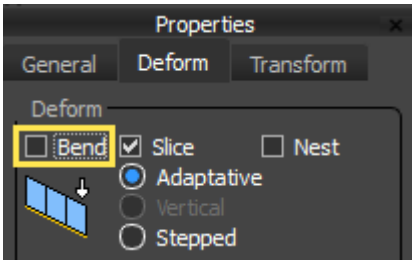
8.2 Turning a Corner Segment's Bend setting On and Off.

If you look at the poles you'll notice that they are deforming unnaturally.

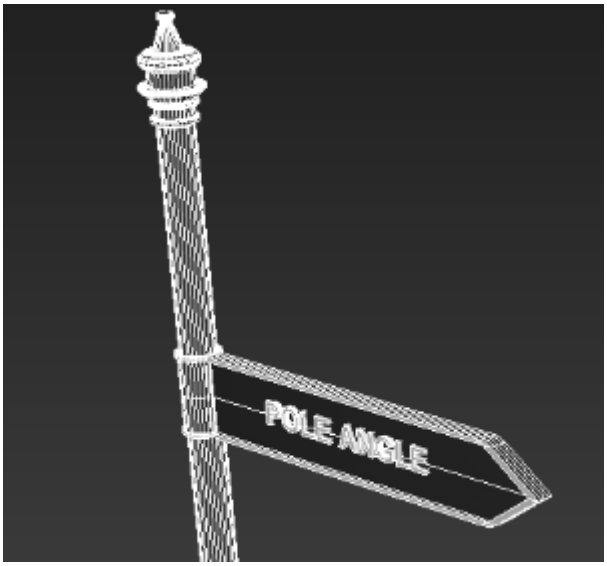


This is because RailClone's default behaviour is to try and bend the segments to follow the path. On a hard corner this can cause some issue so in many cases it is desireable to turn off the corner segment's bend setting. To do this follow these steps:

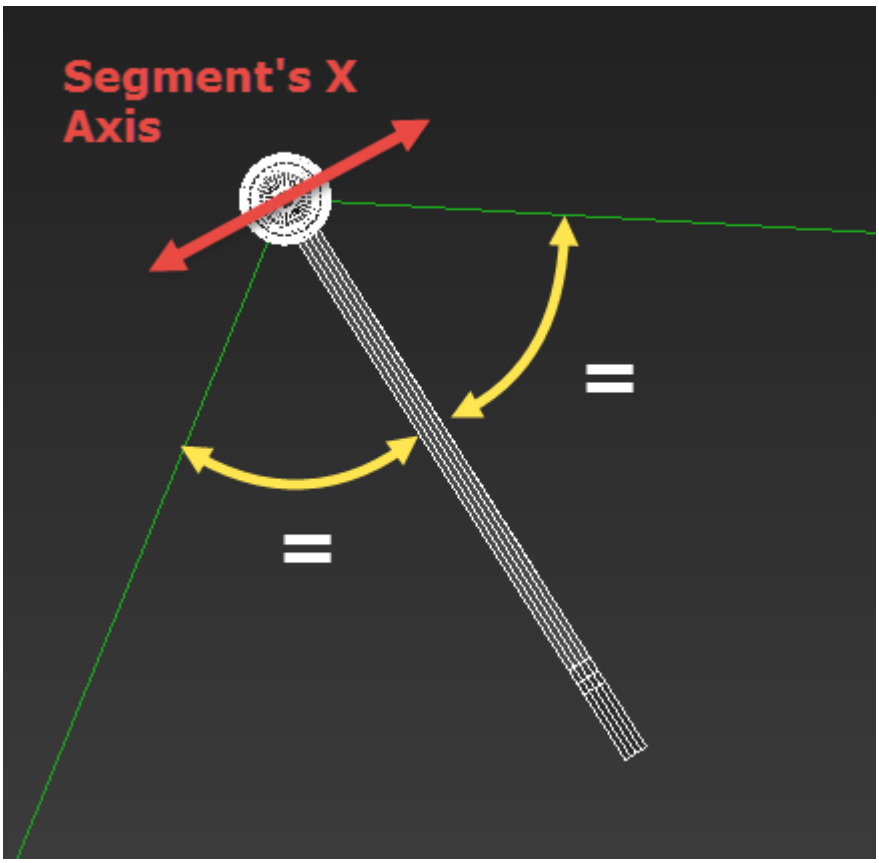
1. Select the *Post* segment
2. Select the **Deform** tab in the Segment's Properties
3. Turn off **Bend**.



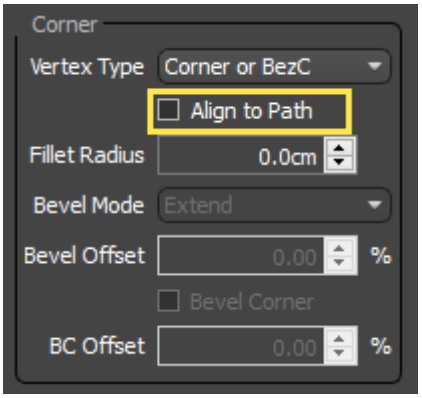
4. With Bend turned off the segments are no longer deformed to follow the spline and instead the segment rotates to bisect the angle of the spline.



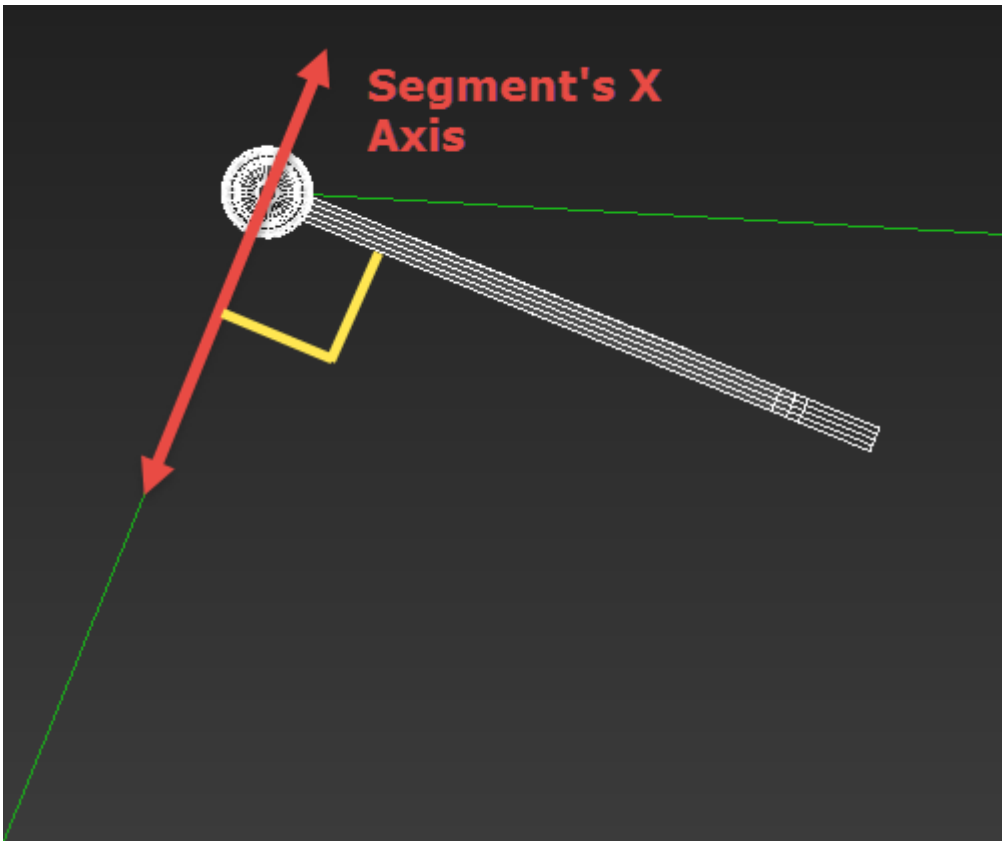
5. If you look at a top view you can see this; remember the sign was originally orientated so that it is pointing at a 90 degree angle from the path.



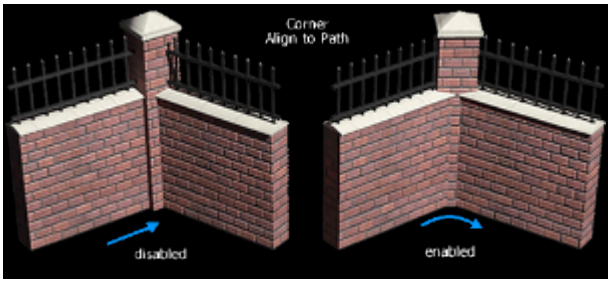
6. If you would prefer that the corner segments did not rotate you also have the option to orientate the segment so that it is aligned with the **preceding** spline segment. To do this, go to the Generator's Corner settings and turn disable **Align to Path**



7. From the top view you will see that the segment's X axis is now in line with the preceding spline segment.



Turning off Align to Path is particularly useful for wall, railing and fence styles to correctly align a corner post that is square in plan. To see an example of this please see our introductory Masonry Wall tutorial.

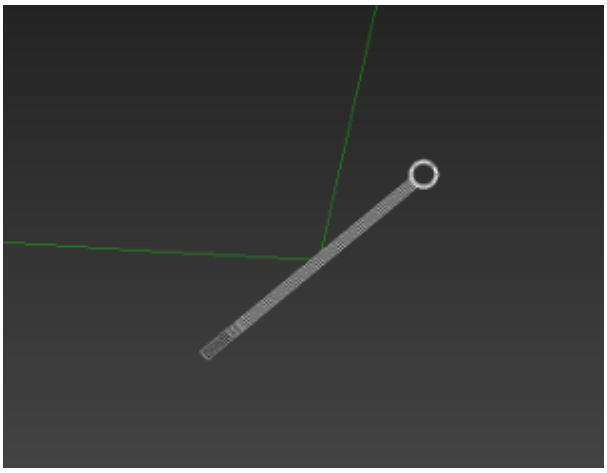


8.3 Using the Compose Operator with Corners

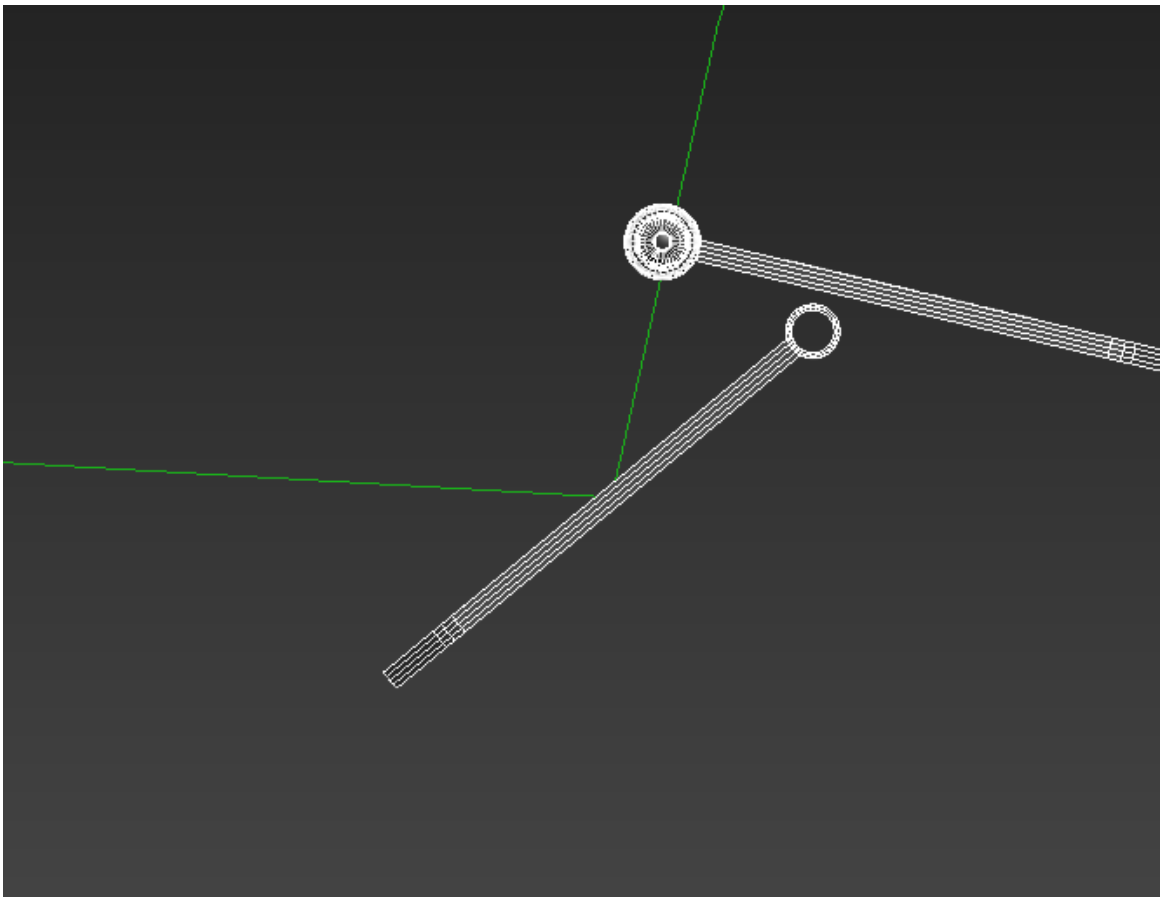
When using a single corner input the segment's geometry is centred on the vertex. However if you use the compose operator with the corner input slot, you will get different behaviour depending on whether you have an odd or even number of segments attached. In the next exercise we'll add the other two signs on either side of the post, and look at the behaviour of the corner as we add additional segments.

Exercise: Composing Corner Segments

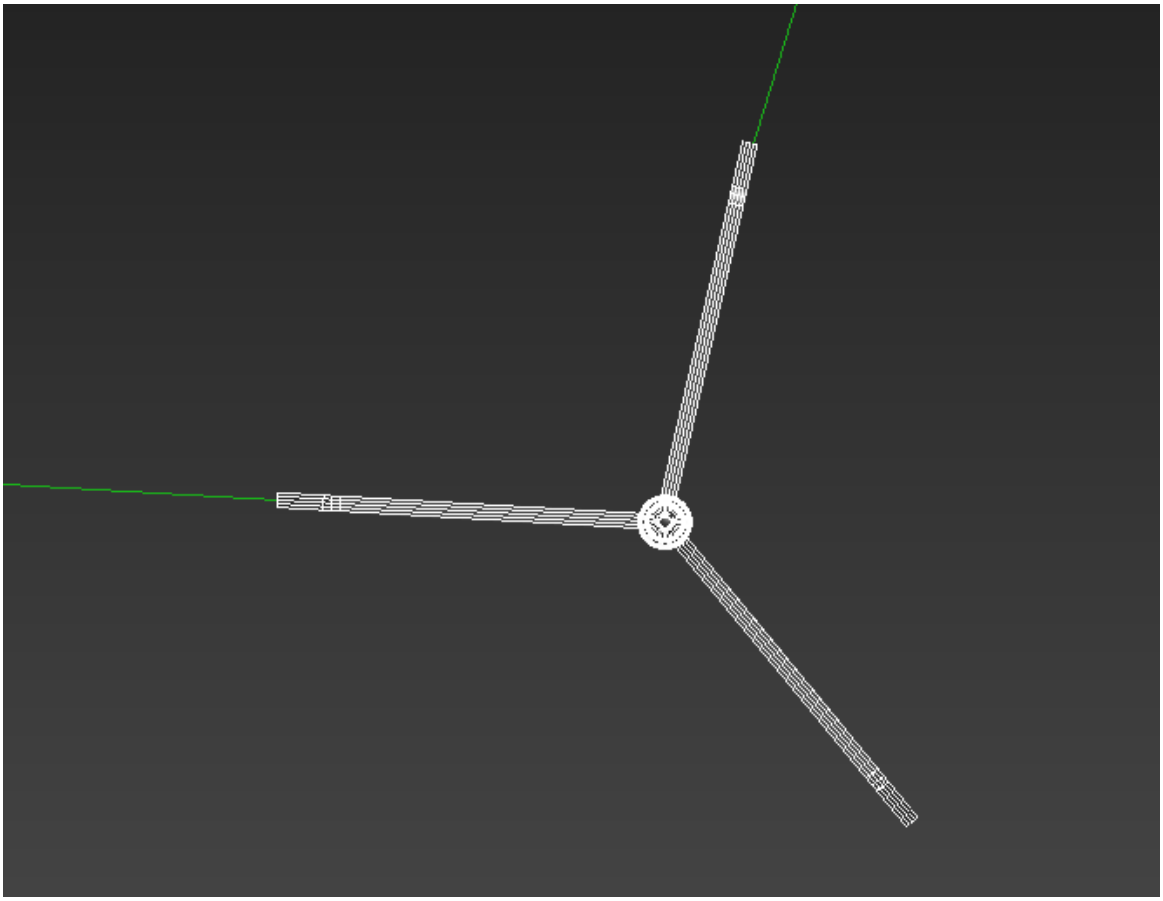
1. Continue from the previous exercise and add a new **Compose** operator to the generator's **Corner** input, replacing the post segment that was already there.
2. Wire the *Start Sign* segment to the Compose operator's **first** input. If you look at the style from the top view you'll see the expected behaviour as the sign is centred on the corner vertex.



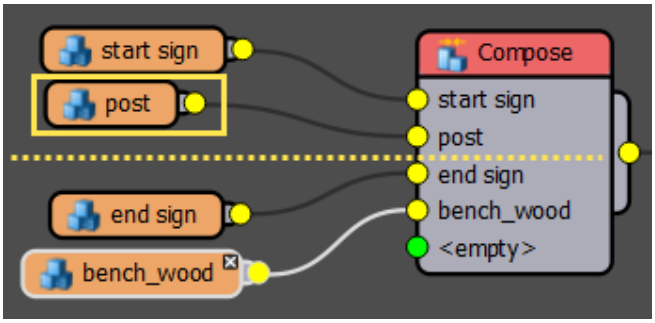
3. Add the Pole segment to the compose operators second input. Instead of centring both segments to the spline, the first segment stays in the same place and the pole is added afterwards. This is because when using an **even** number of segments in the compose operator, RailClone centres the segment immediately before the vertex, then places other segments on the left or right depending on order in the compose operator. This means that using an even number of segments always creates an **asymmetric** corner composition.



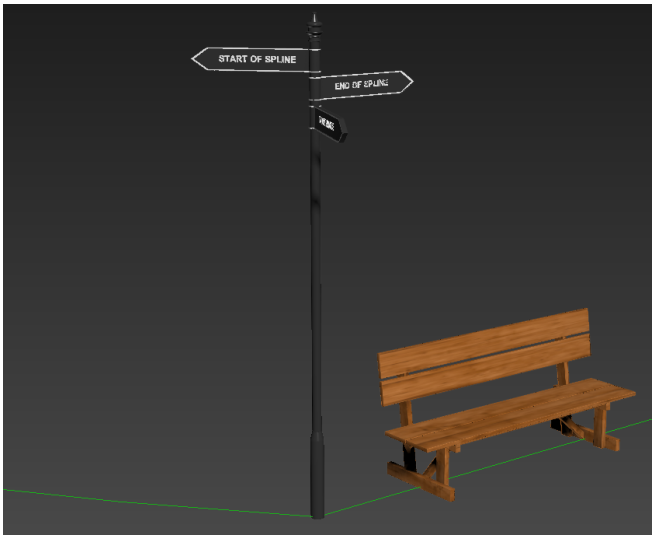
4. Add the *End Sign* segment to the Compose operators 3rd input. You now have and **odd** number of segments and the pole is centred on the vertex. When an odd number of segments is used, the middle segment is centred and the others are placed before and after this depending on the their order in the compose operator. This means that using an odd number of segments always creates a **symmetrical** corner composition.

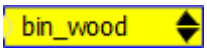


5. Let's keep going. Add the *Bench Wood* segment to the **Compose** operator. A bench is added after the signpost but nothing else moves. This is because this is once again an even number with no centre segment so instead RailClone picks the one preceding the middle, which is the post.

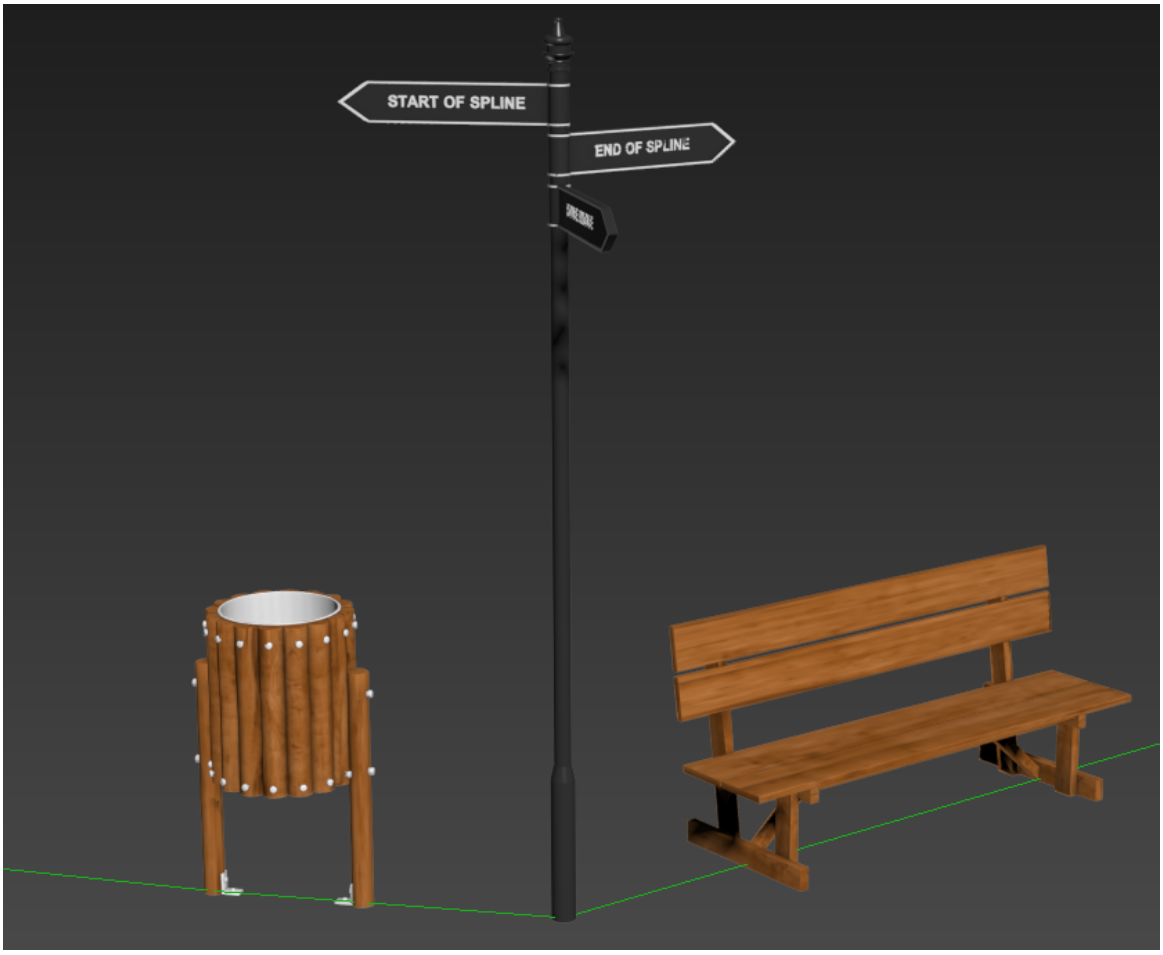


The other segments are placed either side of the post according to their order in the Compose operator



6. Finally we'll add a *bin* to the first input slot of the Compose operator (using the up and down buttons to move it to the correct position - 

). Once again, there are an odd number of segments in the Compose operator so the middle is centred on the vertex. You'll now have a pole on the vertex with a bin and sign before it, and a sign and bench after. You can also see that the Align to Path controls only apply to the segment that is directly located on the vertex, the other segments rotate to follow the spline section they are situated upon.



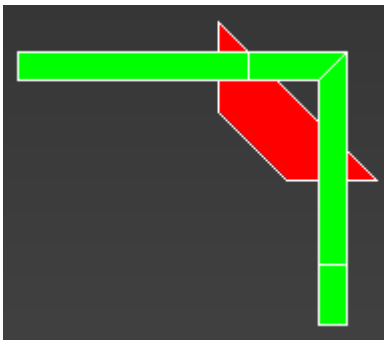
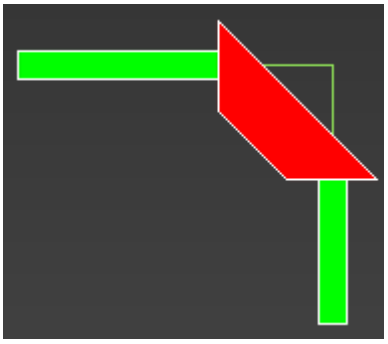
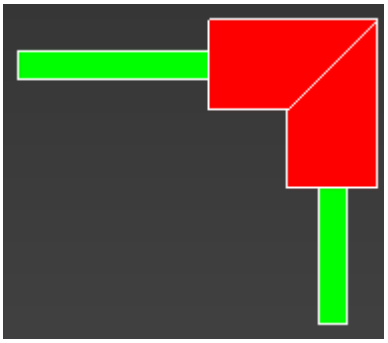
By understanding how the compose operator, Bend and Align to Path work together, you can achieve a high level of control to create sophisticated corner compositions.

8.4 Bevelling Corners

If you need a seamless corner then segments can also be bevelled, slicing the geometry to ensure a perfect mitre. There are actually two sets of bevelling controls in RailClone, one for the default segments and another for the corners themselves. Default segments can be bevelled based on one of three rules.

1. **Reset**: Each segment is placed in its default position, and simply sliced at the corner vertex.
2. **Extend:** Extends the geometry of the segments along the bevel, giving an appearance of continuity around the corner.
3. **Symmetric**: Creates a symmetrical composition with the segment equalised either side of the Corner vertex.

In contrast, Bevelled Corners are simply sliced. The important thing to recognise is that though these two bevel operations have separate controls, they are in fact interrelated. Depending on the combination of their settings you will get different effects.The table below shows the r elationship between the Bevel Mode and Bevel Corner settings.

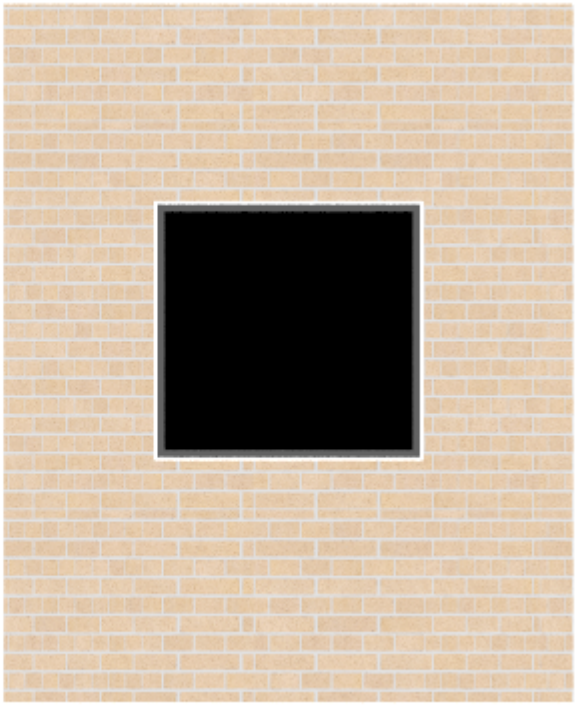
Bevel Mode	**Bevel Corner On/Off**	**Default Segment Behaviour**	
Extend, Symmetry or Reset	Off	Default segments ignore the corner geometry, continue to the corner and are sliced. The corner is deformed to follow the path.	
None	Off	Default segments are sliced by the corner segment. The corner is deformed to follow the path.	
Extend, Symmetry or Reset	On	Default segments stop at the corner segment. The corner is bevelled.	
None	On	Not possible, a bevel mode must be chosen to use bevel corner	

In the examples above we looked at the importance of understanding how the Compose operator works with Corner inputs. When using Bevel mode, more-or-less the same rules apply. The alignment of the geometry in relation to the Corner depends on the number of segments used in the Compose Operator. The illustrations below explain how using a single, odd or even number of segments affects the alignment when using bevel mode.

Segments used in this example:

In the following examples these segments are used in the corner input. Where multiple segments are used, they are wired to a compose operator.

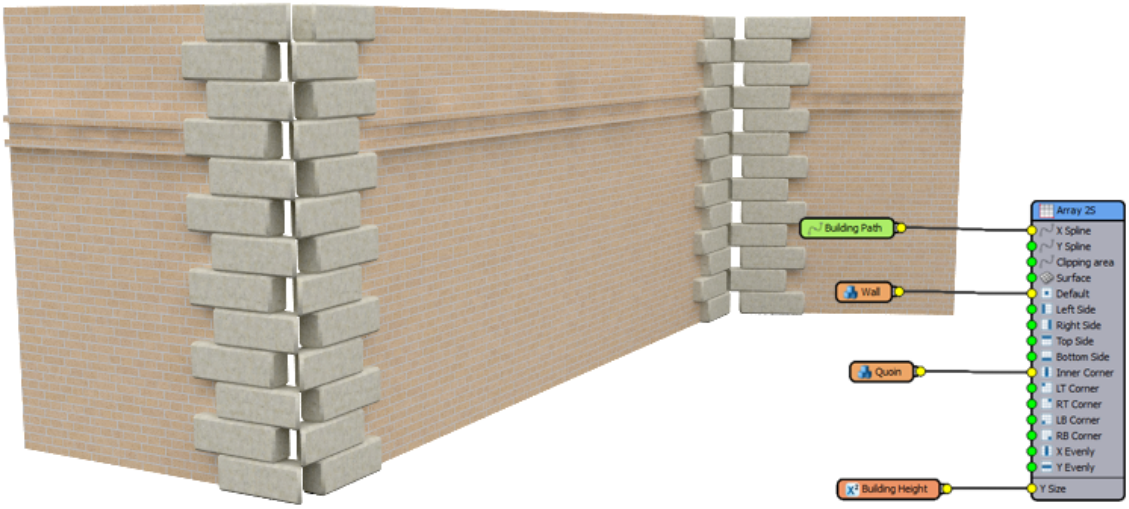
Quoins



Window

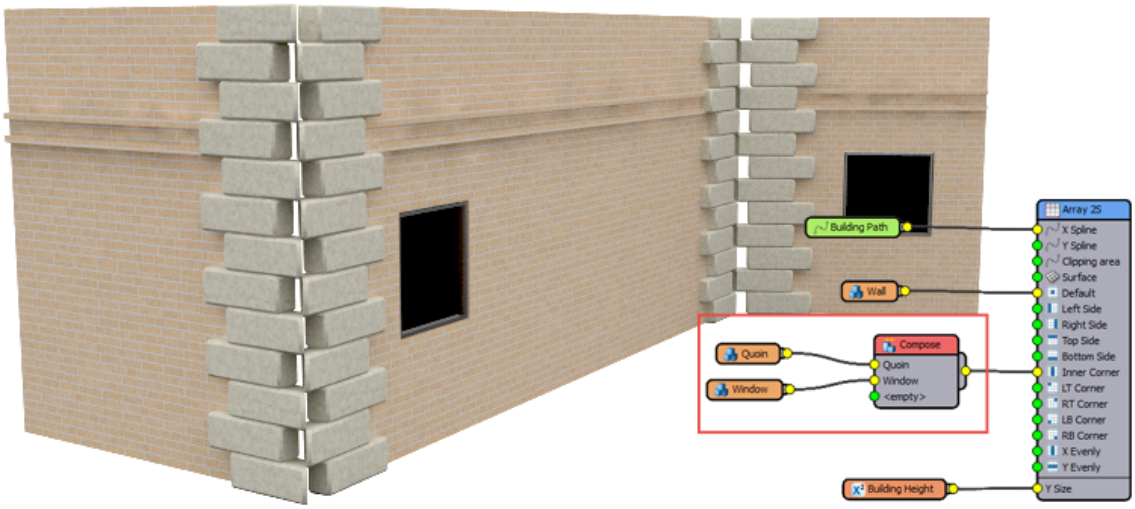
Using a single corner segment

Bevel mode works by duplicating the middle segment either side of the mesh and then slicing the intersection, this means that using a single segment in the corner input looks a little different with bevel on. The position of the pieces is adjusted to ensure that the full length, measured on the outside face and along the X axis, is maintained irrespective of the angle of the corner.



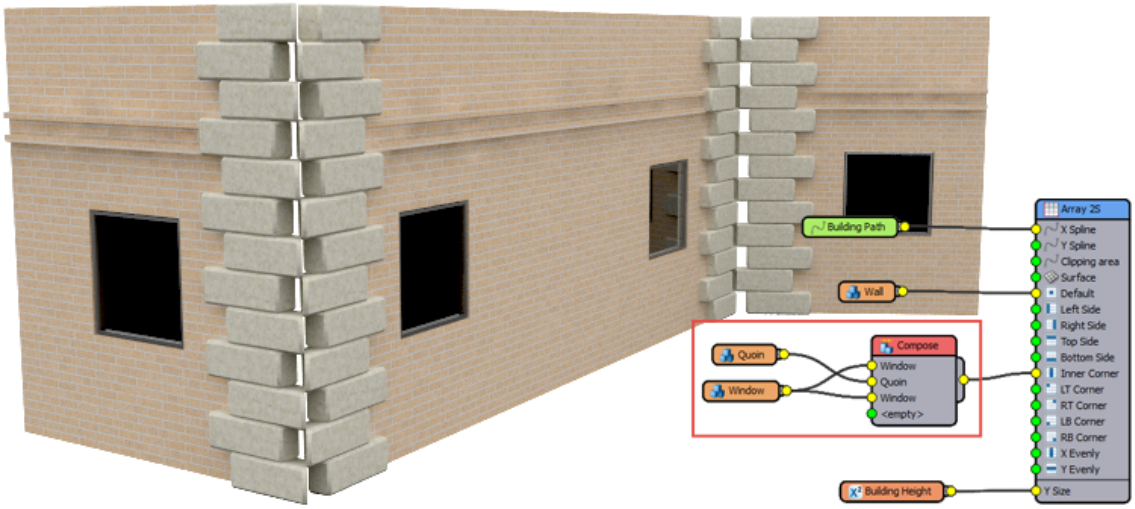
Using an even number of corner segments

When using an even number of segments, RailClone duplicates the segment immediately before the centre then places other segments before or after depending on their order in the Compose operator.



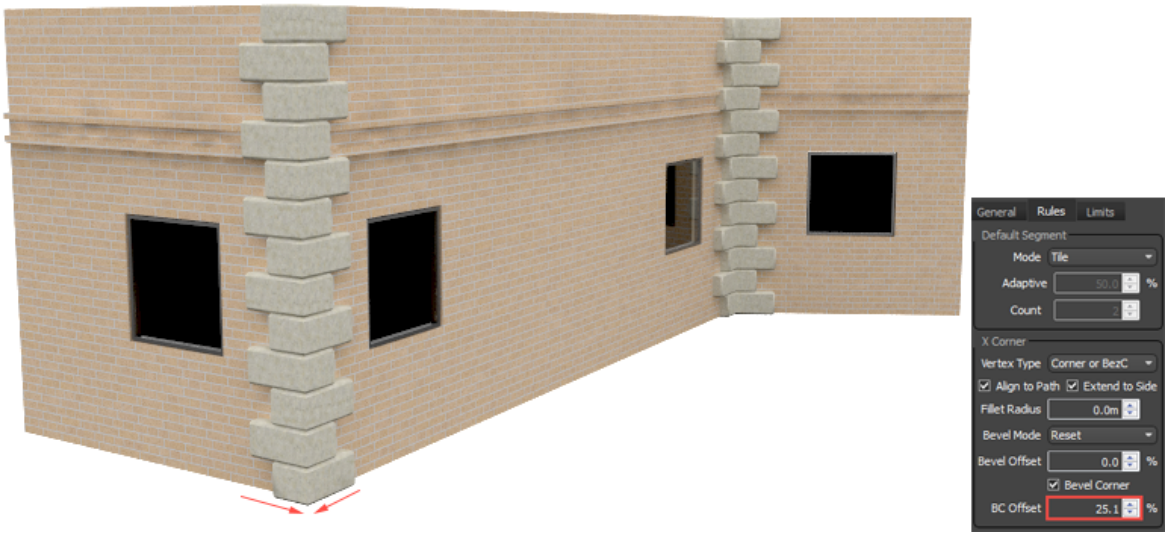
Using an uneven number of corner segments

When using an even number of segments, RailClone duplicates the middle segment then places other segments before and after depending on their order in the Compose operator.



Bevel Corner Offset

As you can see using Compose operators with Bevel Corner is similar to the exercise earlier in this chapter. The only difference is that the segment placed on the vertex is duplicated and bevelled. In the image above you can see a gap caused by the stepped pattern. To solve this you can use the **Bevel Corner Offset** parameter which allows you to adjust the slice point of the Corner segments. Distance is measured as a percentage of the length of the corner segment. Positive values pull the segment towards the corner, negative values will move the segments away from the corner. If a negative value is sufficiently large a gap will appear. The example below illustrates the effect of increasing the Offset value, bringing together and slicing the corner segments to close up the gap.



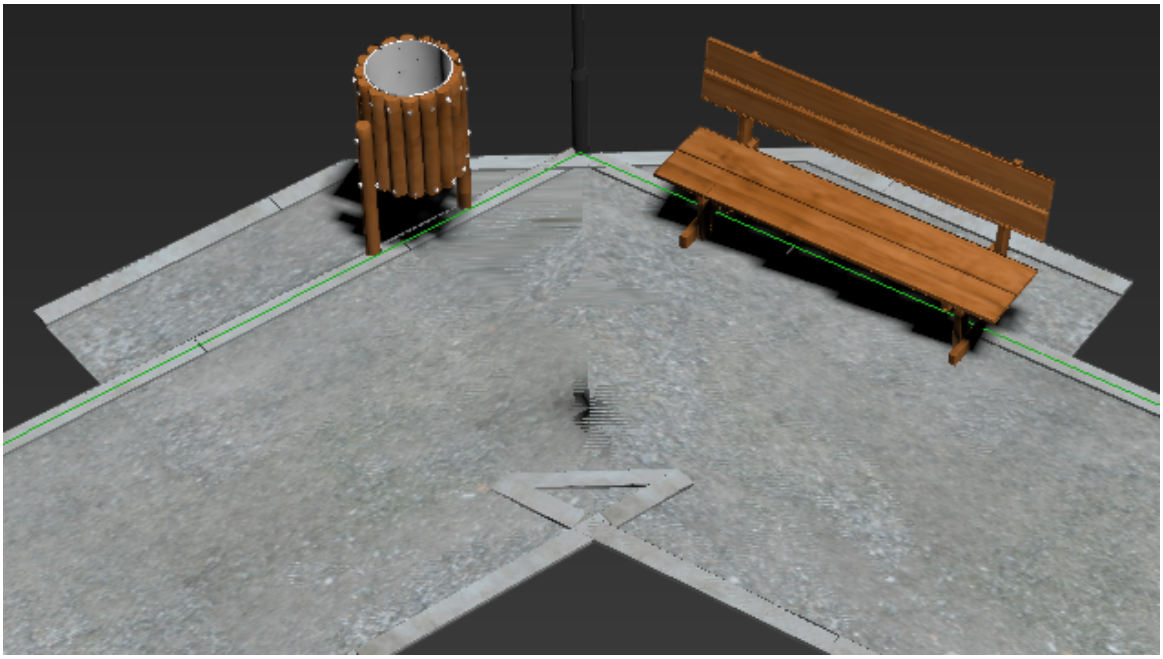
There are a couple of tips to get the most out of the Bevel Corners feature:

- The Bevel feature works better on straight lines and wide corner angles. You may get undesirable results on narrow angles or curved paths.
- When using closed paths, it's not possible to adjust the corner segment's offset on the last corner, because it depends on the path's length.
- Bevel is disabled when the Default Segment Mode is set to Adaptive or Count . We will look at a workaround for this later in the chapter.

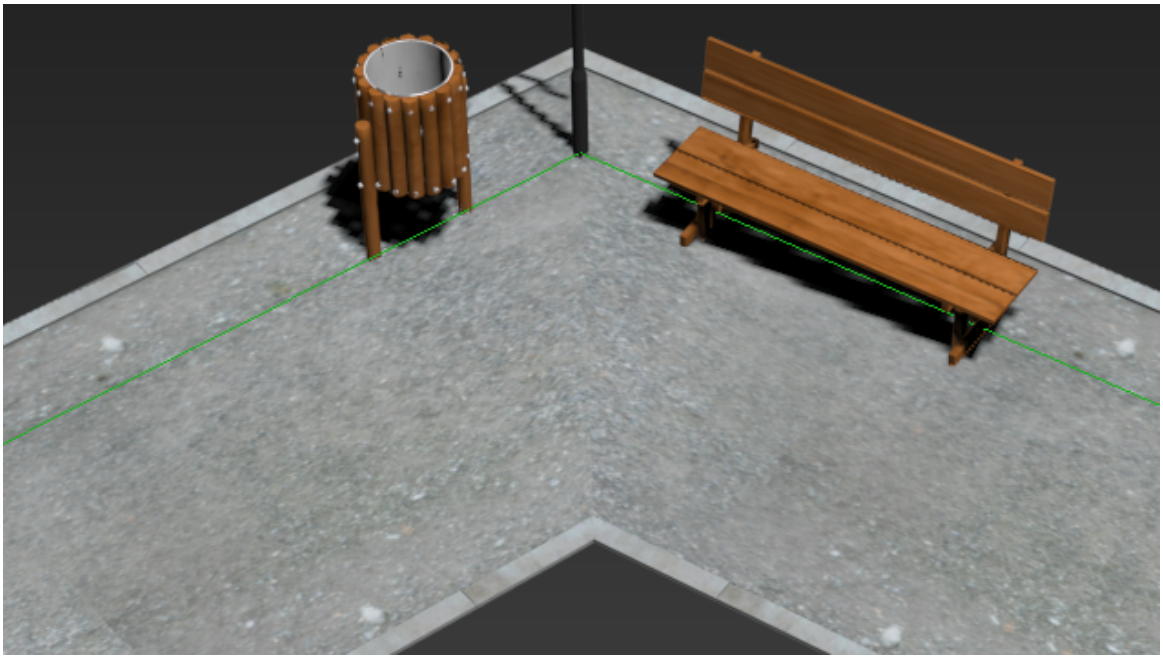
Exercise. Bevelling the Footpath

In this exercise we'll add composed corner segments to the footpath and use bevel to create a hard bend. The Corner segments are slightly wider than the main path to accommodate the bin, bench and signpost.

1. Continue working with *chapter_8_signposting_corner_angles.max* and select the *rc_path* object
2. The Default segment is already wired-up and there are 3 corner segments ready to go. Create a new **Compose** operator and wire this to the **Corner** input
3. Wire the *Path Corner Mid* segment to the **Compose** operator's **first** input. The corners look like a bit of a mess as the segment is deformed around the corner and the default segments continue over the top.



4. To fix this select the Generator and turn on **Rules** > **Corner** > **Bevel Corner**. The corner segment is now correctly bevelled and the default segments are sliced on either side.



5. To add the other sections, wire *path_corner_start* to the **Compose** operator and move it so that is is in the **first** input slot. Then wire *path_corner_end* to the Compose operators **third** input slot. The corner is now complete and is neatly sliced with transitions between the wide and narrow parts of the path.

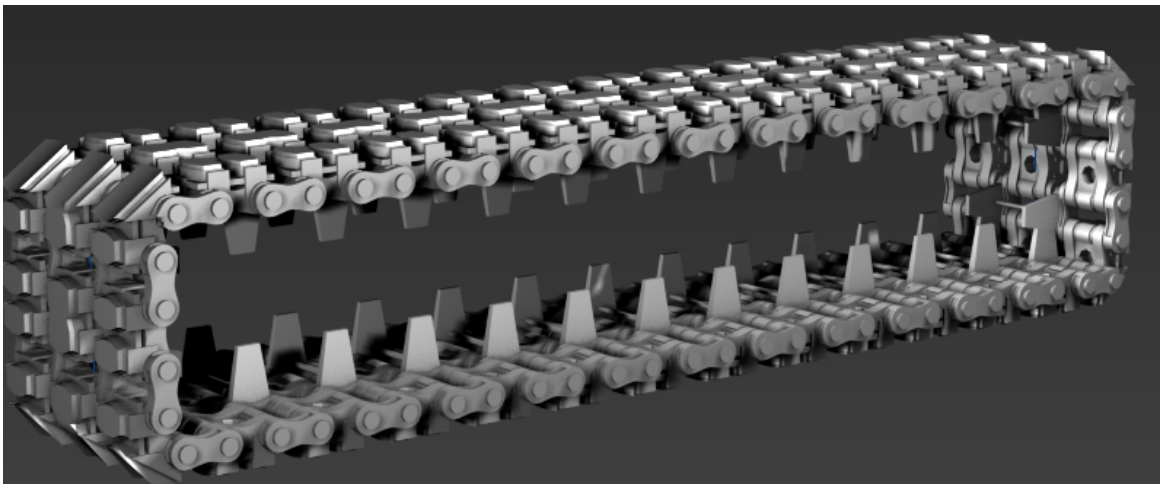
In the next section we'll look at how you can automatically turn hard corners into curves.

8.5 Automatically Create Curves with Fillet Radius

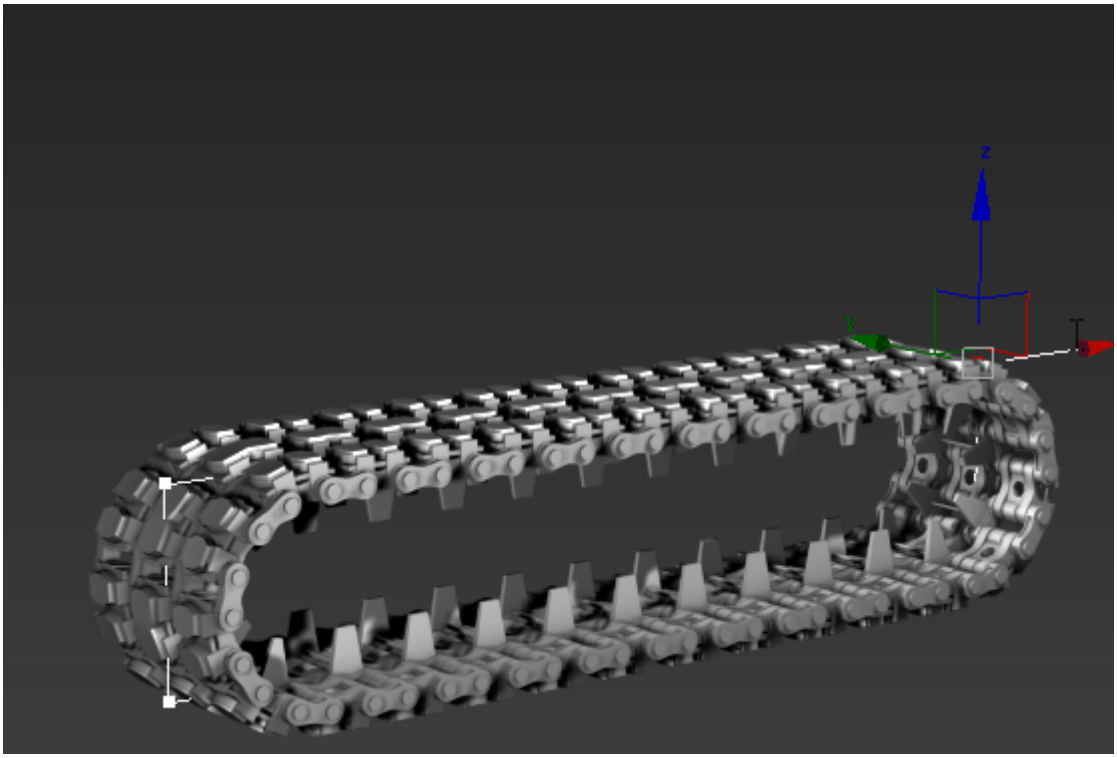
Endlessly filleting a spline corners can be a tedious activity, especially because once you've used Max's built-in fillet tool it's very hard to go back and change the size. RailClone makes this easy by allowing you to automatically adjust the corner vertices on a path to create rounded corners. The best thing is, this is fully parametric so you can adjust it at any time. Here's how this feature works.

Exercise: Using Fillet Radius

1. For this example we'll fillet the corners of a spline that is being used to generate tank tracks. Open *chapter_8_tank_tracks_start.max* from the downloads for this guide. You'll see a track style that is following a rectangular spline.



2. Select the Tracks and open the Style Editor.
3. Select the **Generator**. Go to the **Rules** tab in the properties and enter a value of **200cm** for **Fillet Radius**. In this example I'm using a much larger value than is necessary. RailClone will use this as a maximum value and create the biggest fillet it can up to this size depending on the length of the shortest spline segment.
4. Try changing the size and shaped of the spline. You'll find the fillet automatically curves the corners while the spline itself remains very easy to adjust.

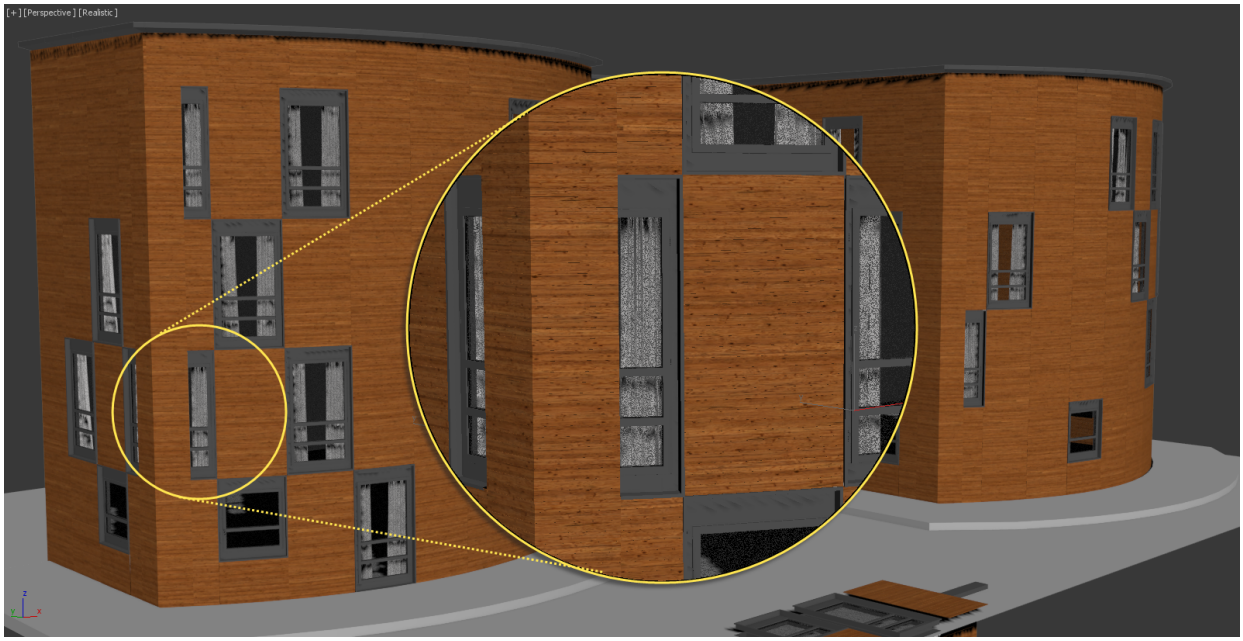


Note that when Fillet Radius is used Bevel Mode, Bevel Offset and Corner Bevel are disabled. This is because the spline is now curved and no longer contains sharp corners, so these settings don't apply.

8.6 Tip: Bevelling Corners in Adaptive or Count mode.

Earlier in this chapter when looking at bevel mode we mentioned that this feature is not available when using **Default>Adaptive** or **Count** mode. Though this is true, there is an easy workaround that allows you to use this two features together. To do this you split the style across two generators, one for the Default segments and a separate generator for the Corners.

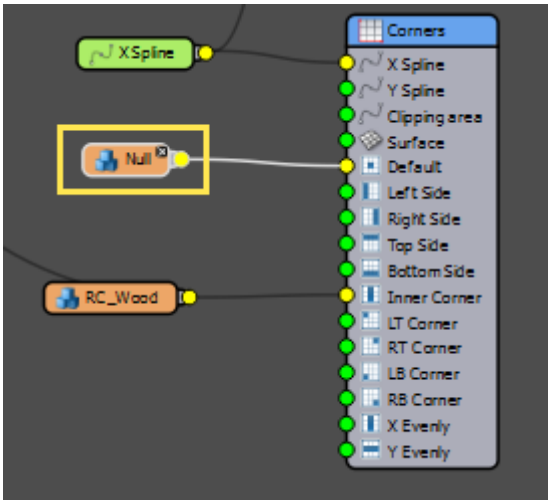
Take this façade for example, we want to bevel the corners but we can't really use Default Tile mode otherwise we'll end up with sliced windows.But if we use Adaptive mode, which would be preferable, we can't get the bevelled corners.



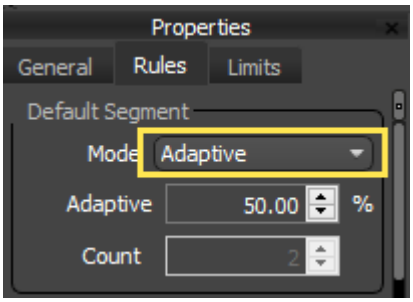
To fix this, separate the style into two generators by following these steps.

1. Open *Chapter_8_adaptive_corners.max* from the downloads for this tutorial.
2. Select the Façade and open the style editor.
3. To save some time we'll duplicate the existing A2S generator. Select the generator node, Right-Click and then select Copy. Right-Click again and select paste. You'll now have two identical generators. (Lite users can duplicate the RailClone object)
4. In the new generator, detach all the inputs with the exception of Inner-Corner. This generator will be used only to create these segments.

5. The A2S generator won't create inner corners unless there is something attached to the default input. So create a new Segment and wire it to the Default Input. There's no need to add any geometry to this node, it's just being used to trick RailClone into creating the Corner segments.



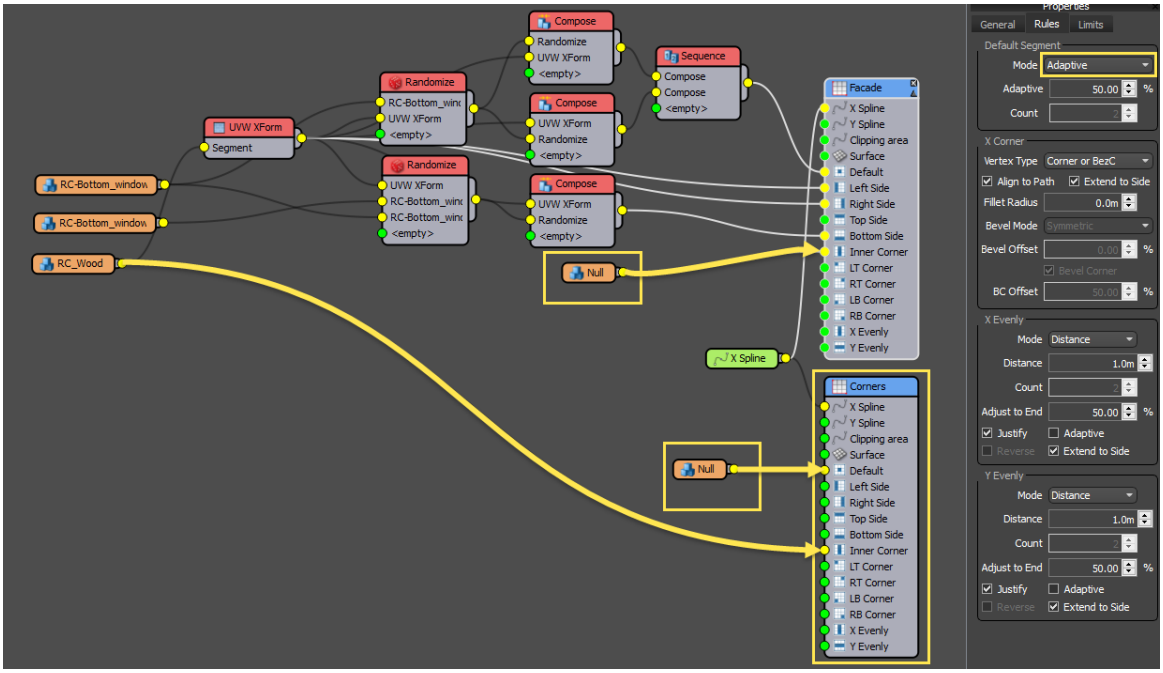
6. Go back to the original Façade generator and change the Default Tiling mode to **Adaptive**.



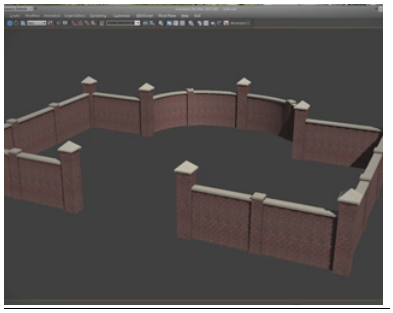
Now none of the the segments are sliced, which is much better.

7. Now we need to create a gap at the corners for this style so that the other generators can fill in. To do this, wire a new **Null** Segment to the **Inner Corner** input, replacing the existing connection.

8. Select the **Null** segment and increase the **Fixed Size > X** setting until the corners and façade align correctly. **1.66m** should do it. You now have the advantages of using adaptive default segments **and** bevelled corners. Here's the final graph, changes have been highlighted in yellow.



8.7 Related Tutorials



Masonry Wall

The Masonry Wall demonstrates how to add posts to a simple wall style and control their rotation so that are correctly oriented for walls with 90 degree bends

Power Lines

Use splines to create power lines with a new pole added on each corner vertex.

Create a seaside promenade

In this tutorial we explain how to use RailClone to create a stone balustrade with posts added at evenly distances and on corners.

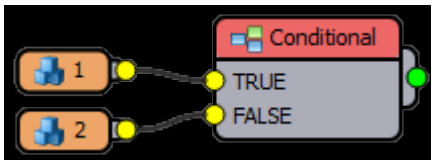
Documentation

For more information please see the 1D arrays - Generator L1S, 2D arrays - Generator A2S, and Segments sections of the online documentation.

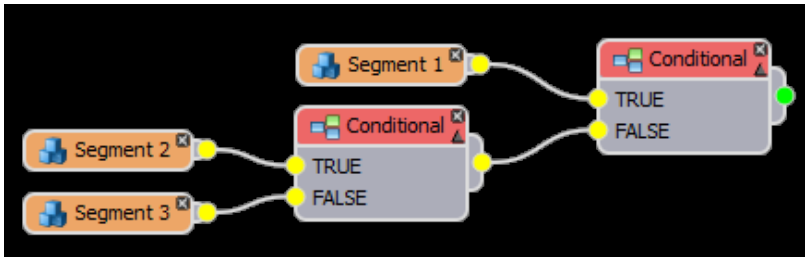
9 9 - Creating Conditional Relationships

Sometimes you may want to swap segments dynamically based on other conditions in the array. For example you could create a truss that increases in depth as the length gets longer, or a corner segment that changes depending on the angle.

Using the Conditional operator you are able to change segments based on spline length, the segment's position in the array, whether the spline segment is a curve or a line, the segment count on the X and Y axis, and the splines segment's material ID. Best of all, the Conditional operator is very easy to use, it takes two segments as inputs and only returns one of them depending on whether the test conditions return true or false.



If you want to test for multiple conditions, nodes can be nested to create a more complex decision tree. There's no limit on the number of conditional operators that can be connected in a chain, so complex relationships are easy to set-up and use.



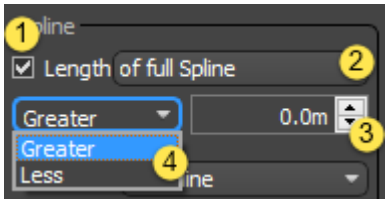
In this chapter we'll look at how to use the Conditional Operator's parameters using a Ducting example. To get started, load *chapter_9_ducting_start.max* from the downloads for this tutorial.

9.1 Spline based Conditions

The first 4 exercise explore the ways in which the conditional operator can be used to test attributes of the spline path.

Exercise: Creating conditions based on spline length

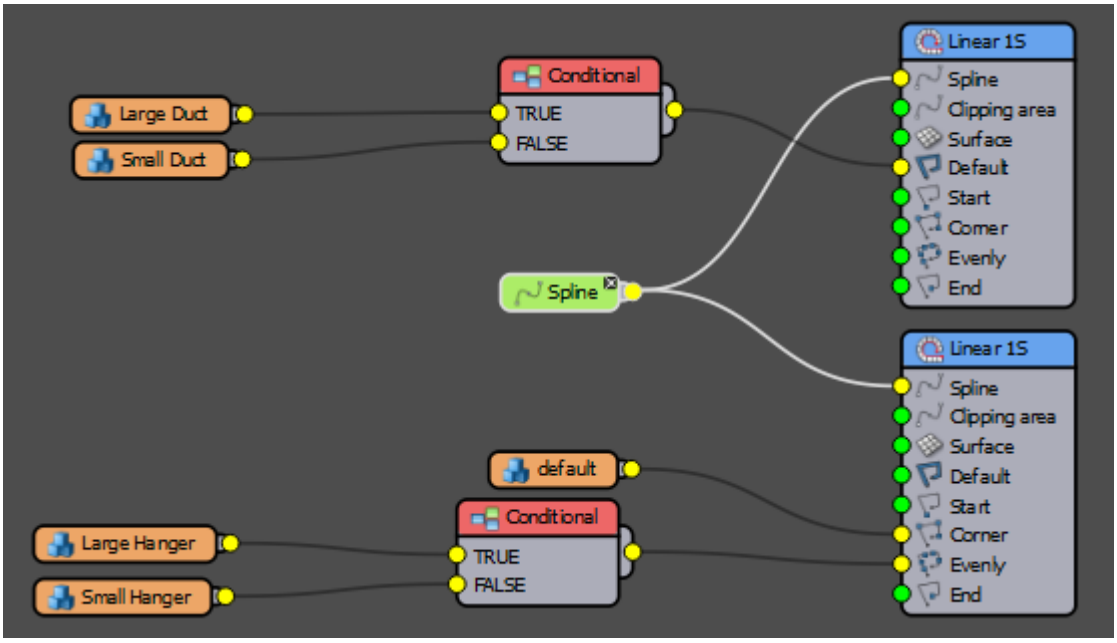
Using the Conditional operator you can use different segments based on the length of the spline or spline section. Though this is only a small part of what this operator is capable of, even this small subsection of conditional rules provides a lot of options. The following four steps demonstrates a simplified generic procedure for creating conditional setups using the length parameter.



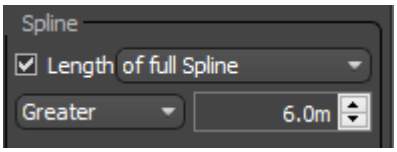
1. First of all, you need to turn on the condition which you want to test. Only one can be activated per Conditional node, so if you want to test more than one condition you'll need to connect multiple nodes as shown above.
2. Choose whether you want to check the length of the **full spline** or just the length of the current **spline segment**.
3. Enter a **Length**, measured in scene units.
4. Choose whether you want to return True if the current spline or segment length is **greater** or **less than** the length value.

In the first ducting example, we'll look at how to change the size of the pipe based on the length of the spline.The scene has been set up with all the necessary segments, all we need to do is add and configure the conditional operators.

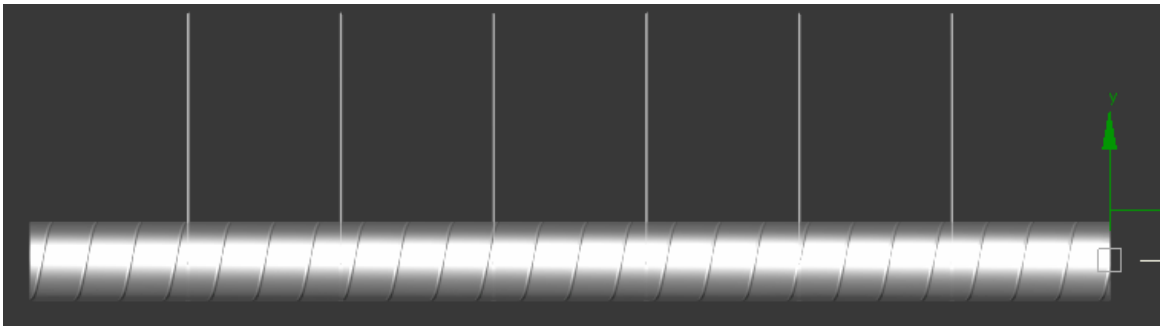
1. Select *rc_spline_length* and open the Style Editor
2. There are two Generators, one for the pipes and another for the hangers. Each of these has a small and a large size segment. For each generator, create a new **Conditional** operator and wire the **Large** segment to the **True** input.
3. Connect the **Small** segments to the **Conditional** operators **False** input.
4. Connect the Duct's **Conditional** operator to its **Generator's Default** input and the Hangers **Conditional** operator to its **Generator's Evenly** input. At this point the graph should look like this:



5. For both Conditional operators, turn on **Spline** > **Length** from the **Properties** panel
6. Enter a **Length** value of **6m**, and change the dropdown list to **Greater**.

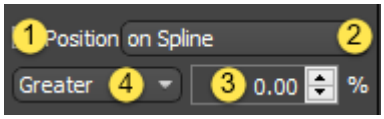


7. These conditional operators will not return the large version of the ducting if the spline is greater than 6m, and the smaller version if it is less than that.



Exercise: Creating conditions based on the segment's Position on the X or Y spline

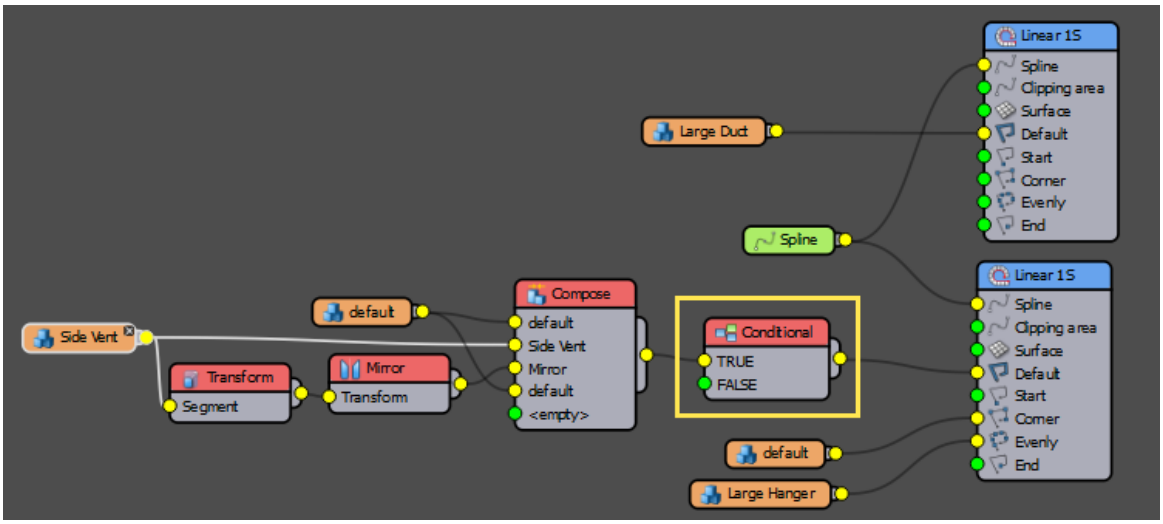
As well as being able to test the overall length of the spline or spline segment, you can also use the Conditional operator to test a Segment's position on the spline or spline section. To use this feature you can follow these steps.



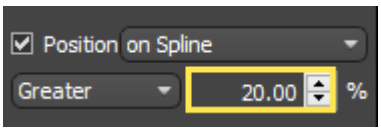
1. Turn on **Properties** > **Spline** > **Position**. Make sure no other checkboxes are ticked.
2. Choose whether you want to check the length of the **entire spline** or just the length of the **current spline segment**.
3. Enter a **position** value to test for, this is measured as a percentage of the current spline or spline section's length.
4. Choose whether you want to return True if the current spline or segment length is **greater** or **less** than the position value.

To see this in action, in our ducting example we'll add some side vents but prevent them from appearing in the first and last 20% of the path.

1. Select *rc_spline position* and open the style editor.
2. This example features two generators. One for the ducting, and a second one for the vents and the hangers. In this exercise we only need to worry about the vents which are wired to the second generator's Default Input.
3. Wire a new **Conditional** operator between the existing **Compose** operator and the Generator's **Default** input. Make sure the **Compose** operator is wired to the **Conditional** operator's **True** input.

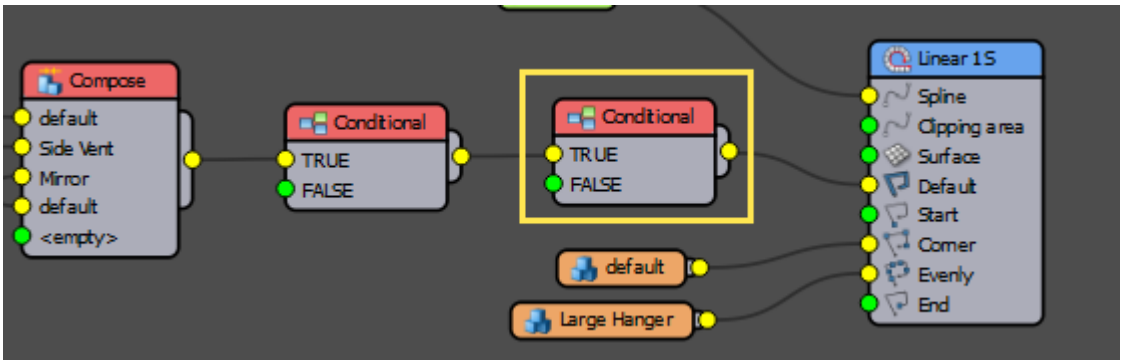


4. Select the **Conditional** operator and turn on **Properties** > **Spline** > **Position**.
5. Set the Percentage value to **20%**. Leave the other defaults as they are.

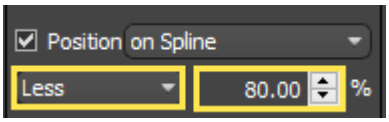


This will prevent the vent from appearing within the first fifth of the spline. Next lets prevent them from appearing in the final fifth by connecting multiple Conditional nodes in sequence.

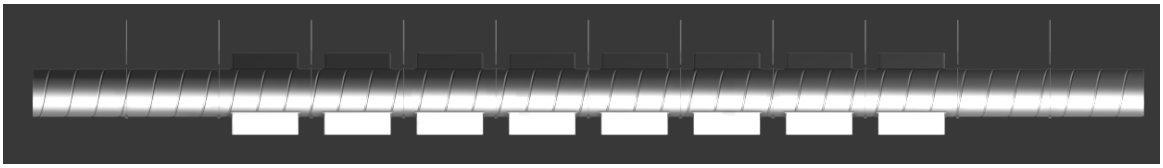
6. Create a new **Conditional** node and wire it between the existing Conditional node and the Generator's default input. Make sure to connect the nodes via the **True** input.



7. Select the **new Conditional** node and turn on **Properties** > **Spline** > **Position**.
8. Set the **Percentage** value to **80%**.
9. Change the dropdown menu to **Less**

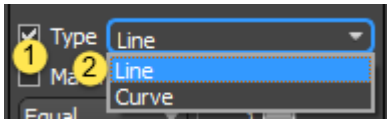


This conditional operator will only return true if the current segment is within the first 80% of the spline's length. When combined with the first Conditional operator we will now only get a vent segment if it falls within 20% - 80% of the spline's length.



Exercise: Creating conditions based on the type of spline segment.

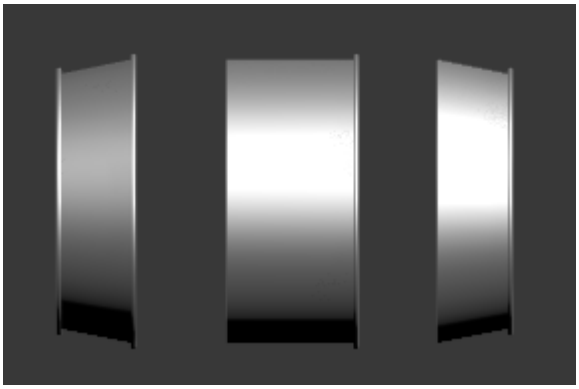
In addition to the length of the spline and a segment's position along the path, RailClone can also test to see if the current spline section is a straight line or a curve. This can be extremely useful if you want to optimise your models as you can use a higher resolution mesh on curved paths so that it bends smoothly, and a lower resolution on straight paths where no deformation is required. To use this feature, follow these steps:



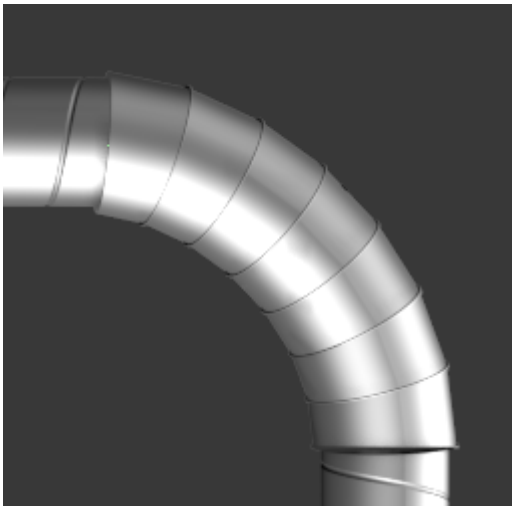
1. Turn on **Properties** > **Spline** > **Type**. Make sure no other checkboxes are ticked.
2. Select the type of spline segment that will return a True result from the drop down list.

In our ducting example, we'll use this feature to create an articulated section that is used only on bends, with a transition at either end to compensate for the slight change in diameter. Here's how

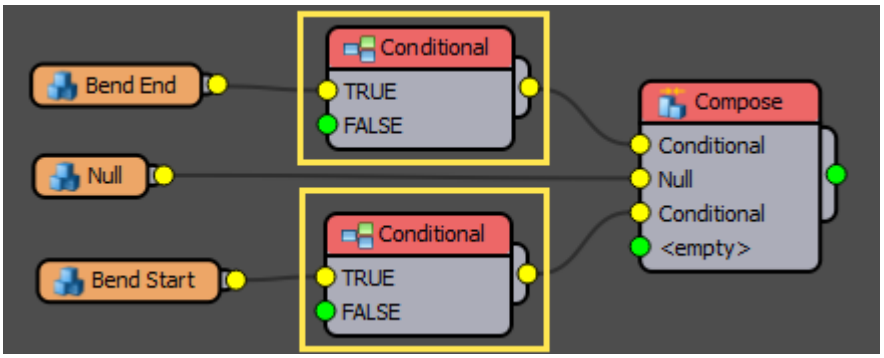
1. Select *rc_spline_type* and open the Style Editor.
2. As in the previous examples, this style features two generators. The first is used for the ducting, and the second one is used on for the hangers. In this example we only need to worry about the first generator. There are also 3 additional segments for the curved sections of the spline. These are called *bend start*, *bend mid*, and *bend end*.



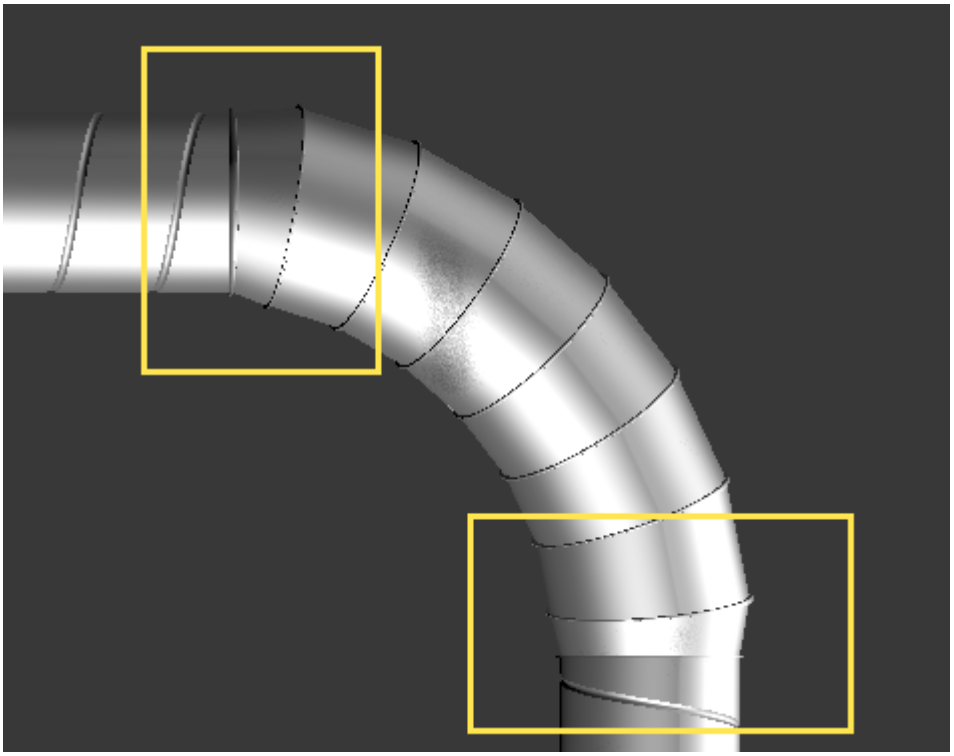
3. Let's start with the Default segments. Create a new **Conditional** operator and wire the *large duct* segment to the **True** input. Wire the condition operator to the Generator's **Default** input.
4. Wire the *bend mid* segment to the **False** input.
5. Select the conditional operator and turn on **Properties** > **Spline** > **Type**. Change the Type drop-down to **Line**. You will now have a different segment on the curved part of the spline.



6. In the next steps we'll add a transitional segment between the straight and curved sections using the Generator's Corner input. When using Conditional operators with the Corner input, an important thing to understand is that the corner uses the spline section that **precedes** the vertex to determine the type. So in the corner pictured above when viewed from left to right, the first corner is of the type Line, because a straight section immediately precedes it, whereas the second corner is of the type Curve, because it is preceded by the arced section of the spline. We can use this to our advantage to create intelligent corners that add a different segment when they are at the beginning and end of a curved or straight section of spline.
7. The existing style has a compose operator for the corner segment. Add two new conditional nodes, each between the existing segments and the compose operator.

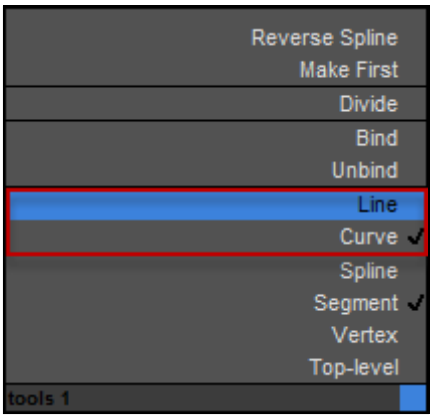


8. Select the Top conditional operator and turn on **Properties** > **Spline** > **Type**. Change the type to **Curve**, this will ensure that this segment only appears on the **second** corner.
9. Select the bottom conditional operator and turn on **Properties** > **Spline** > **Type**. Change the type to **Line**, this will ensure that this segment only appears on the **first** corner.
10. Connect the **Compose** operator to the **Generator's Default input**. You will now have a section with a ramped transition at the corners, between the two sizes of default segment.



Setting spline segment types.

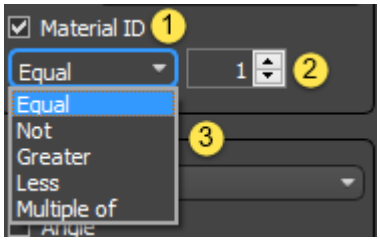
Whether or not a spline section is considered a line or a curve is not determined automatically, it must be set manually by the user. By default most spline objects in Max are created with segments set to the type "Curve". To set the spline segment type property on a spline used as a base object, it is necesary go to the **segment** sub objects of an editable spline, choose a segment, right click to set as either a Line or Curve.



Right click to set the segment type of a spline

Exercise: Creating conditions based on Spline Segment's Material ID

A great way to control which segments are used on a spline is by using Material IDs. Editable Spline objects can have material IDs applied to their segments that can be used in RailClone with the Conditional operator to create a number on interesting effects.To use material IDs you will need to do the following:

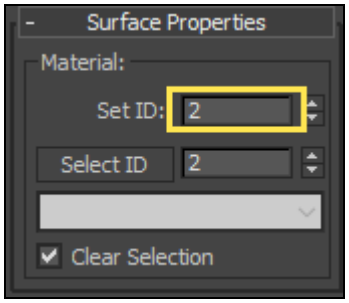


1. Turn on **Properties** > **Spline** > **Material ID**.
2. Choose a Material ID.
3. Choose whether you want to return True if the current spline section has a material id that is equal, not, greater than, less than, a multiple of the Material ID value.

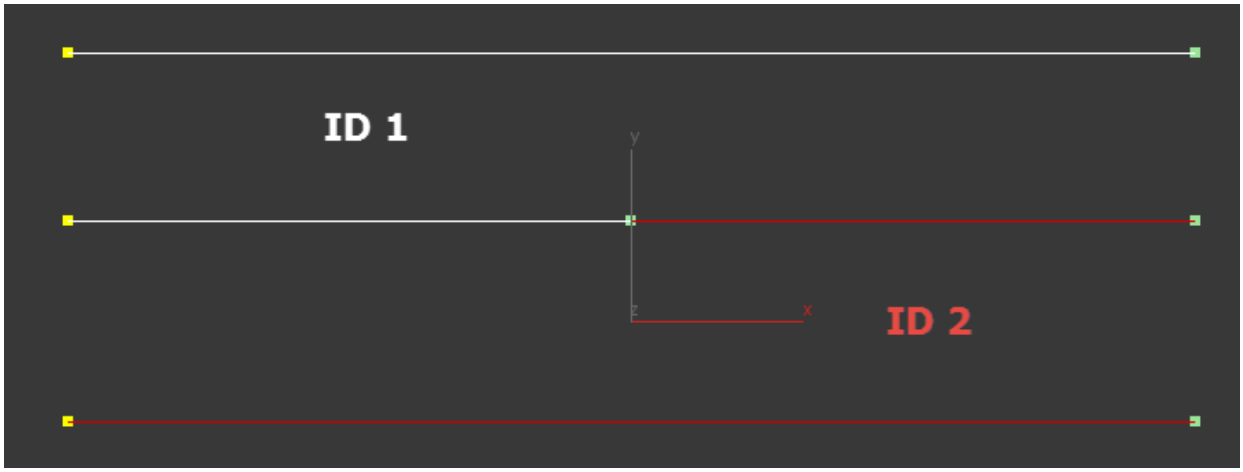
Setting Material IDs

To set the material IDs on a spline follow these steps.

1. Change the object to and Editable Spline or add an Edit Spline modifier.
2. Go to segment of spline sub-object level and select the parts of the object you wish to change.
3. Go to the **Surface Properties** rollout and enter a value for the **Set ID** parameter

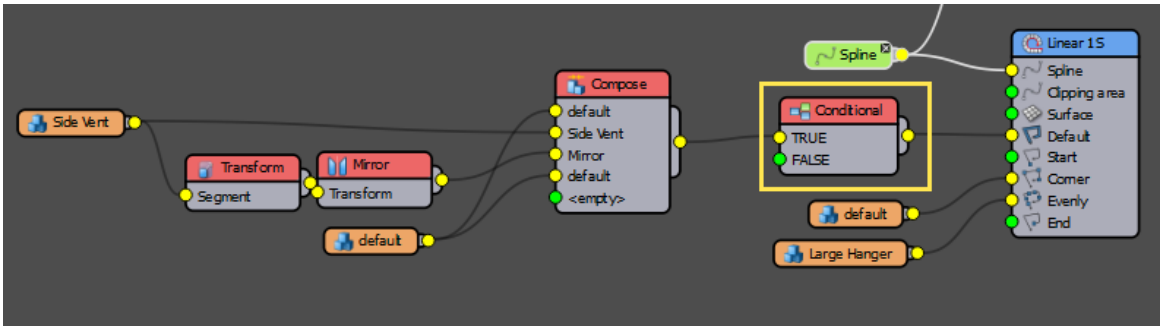


In the example file, the base spline being used by *rc_mat_id_conditional* is divided into two material ids as shown below.

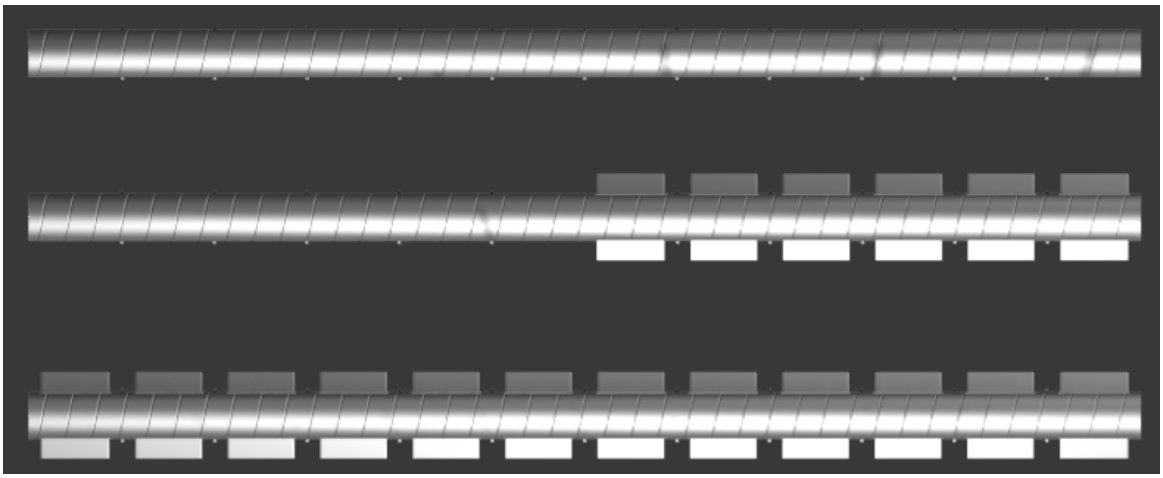


We can use this so that vents are only applied to the spline segments with an ID of 2 using the Conditional operator. Here's how:

1. Select *rc_mat_id_conditional* and open the Style Editor
2. Create a new **Conditional** operator between the existing Compose node and the second Generator's default input.



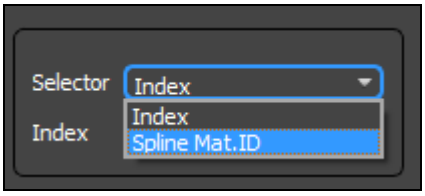
3. Turn on **Properties** > **Spline** > **Material ID**.
4. Set the **Material ID** value to **2**. You will now have vents only on the spline segments with the same ID.



Other techniques for creating conditional rules based on material IDs

Using the Selector Operator

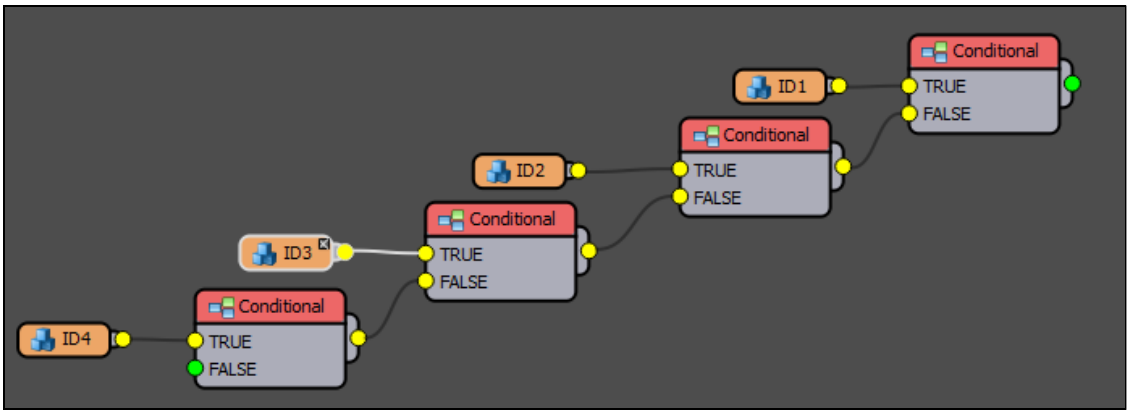
This is not the only way you can create conditional relationships based on material IDs. The Selector operator has a mode that allows you derive the index value from the spline's material ID. To activate it just change the **Selector** dropdown to **Material ID.**



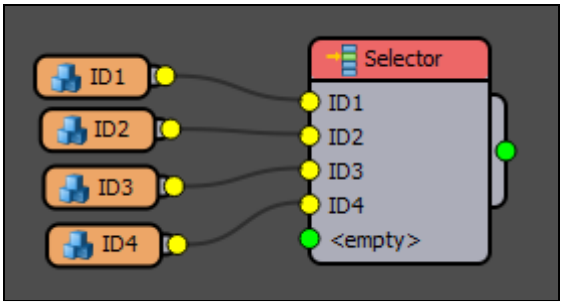
To set up the previous example using a Selector operator instead of a Conditional, try the following:

1. Select *rc_mat_id_selector* and open the Style Editor
2. Create a new **Selector** operator and wire it to the Generator's Default input.
3. Wire a new **Empty Segment** to the **Selector** nodes's **first** input.
4. Wire the **Compose** node to the **Selector** node's **second** input
5. Pick the Selector node and change **Properties** > **Index** to **Spline Mat. ID**.

You will now have the same result as though you'd used the conditional operator. In this example there is little advantage to using this technique, but in the event of using multiple IDs the Selector operator can create the same effect as the conditional operator but with fewer nodes. For example, if you wish to use four segments and apply them to only 1 of 4 material IDs, the conditional node tree would look like this.



However, using the Selector Operator in Material ID mode it would be much simpler:

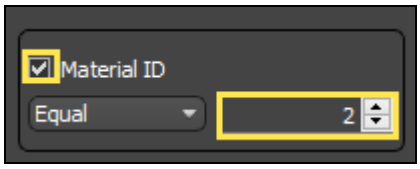


Using a Generator's Limit by Material ID features

As well as the conditional operator, it is also possible to limit the whole generator to use sections of the spline path based on the Material ID. To do this you use the Generator's Limit By Material ID options. This tells the generator to only create geometry if the spline section matches or is less than, or greater than an ID value. It is slightly different from the previous two techniques because each isolated section with a continuous unbroken ID is treated like a separate spline with its own Start and End segments.

This can be useful for breaking a continuous spline into sections, each with the possibility of using the full range of a generator's inputs. You can see an example of this in the Drop Kerb tutorial. In our ducting example we'll just create the same affect as the Selector and Conditional operator shown above.

1. Select *rc_mat_id_gen_limits* and open the Style Editor.
2. Select the second Generator and turn on **Properties** > **Limits** > **Material ID**.
3. Set the **Material ID** value to **2**.



The vents will only be generated on spline sections with an ID of 2.

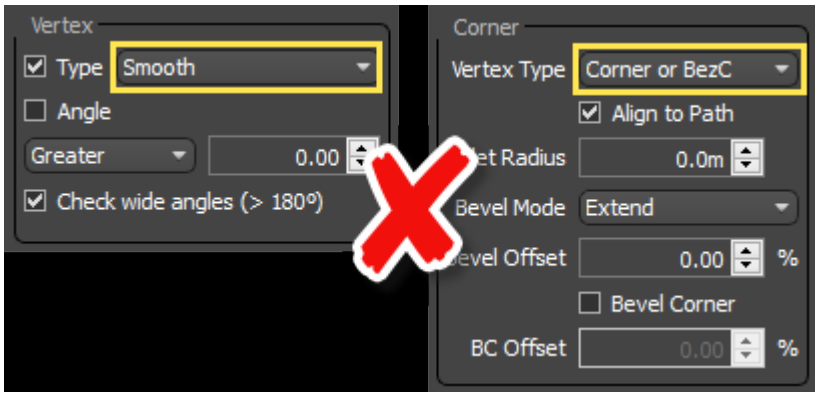
9.2 Vertex based conditions

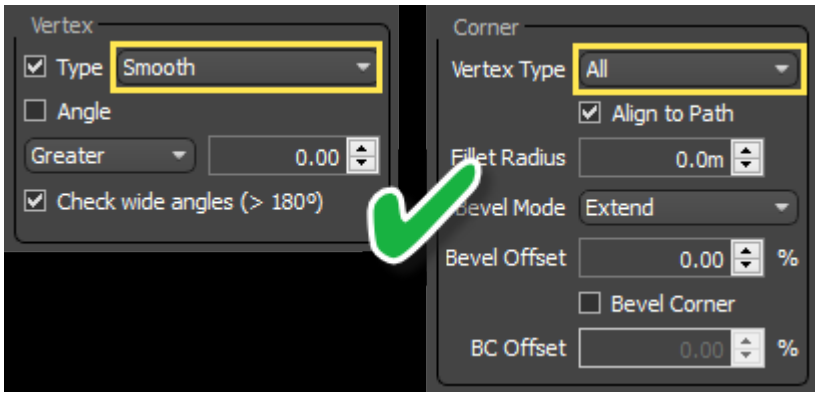
The next 2 exercises explore the ways in which the conditional operator can be used to test attributes of the spline's vertices.

Exercise: Creating conditions based on the type of vertex

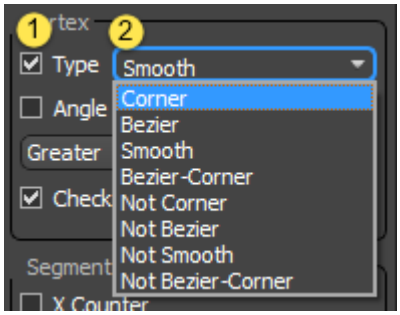
Using the Conditional operator, you can use the type of vertex to select different segments. In this mode, the Conditional operator will return the segment wired to the true input if the vertex matches a specified type, selected from Corner, Bezier, Bezier-Corner or Smooth.

Of course for this to work you will also need to configure the generator to ensure that the specified vertex type will actually generate a corner segment. For example, if you have the Conditional operator to test for a Smooth vertex type, but the generator's corner settings are set to Corner or Bezier-Corner then it won't work because a corner segment will never be generated on a smooth vertex. Instead you would need to change the Generator's Corner type setting to **All** to include vertices of the type smooth (as well as bezier, bezier-corner, and corner).





To use the Conditional operator to test for vertex types:

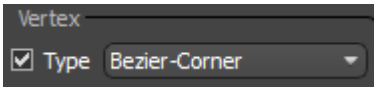


1. Turn on **Properties** > **Vertex** > **Type**

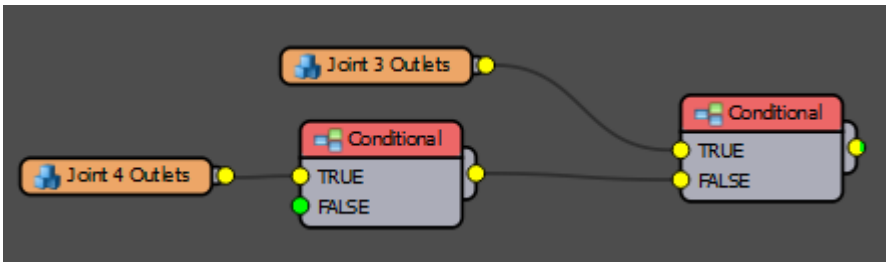
2. Select the type of vertex to return true from the following settings: Corner, Bezier, Smooth, Bezier-Corner. Alternatively you can return true if the vertex is **Not** one of these 4 types.
3. Ensure that the Vertex Type in the generator's properties includes the vertex type you are testing for.

In our ducting exercise we will use vertex types to place 3 different types of joint, a branch right, branch left and an intersection. Because there are multiple conditions to test, we'll have to create a chain of Conditional nodes, each one testing for a different vertex type. To do this, follow these steps.

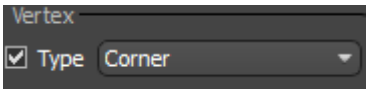
1. Select *rc_vertex* and open the Style Editor. There are 2 corner segments already in the style that need to be wired to the first generator.
2. Create a new **Conditional** operator. Wire the *Joint 4 Outlets* segment to the Conditional operator's **True** input.
3. Select the Conditional operator and turn on **Properties** > **Vertex** > **Type**. Select **Bezier-Corner** from the dropdown menu.



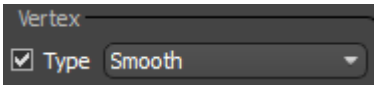
4. Create a second Conditional operator and wire the first one to its **False** input. Wire the *Joint 3 Outlets* segment to its **True** input.



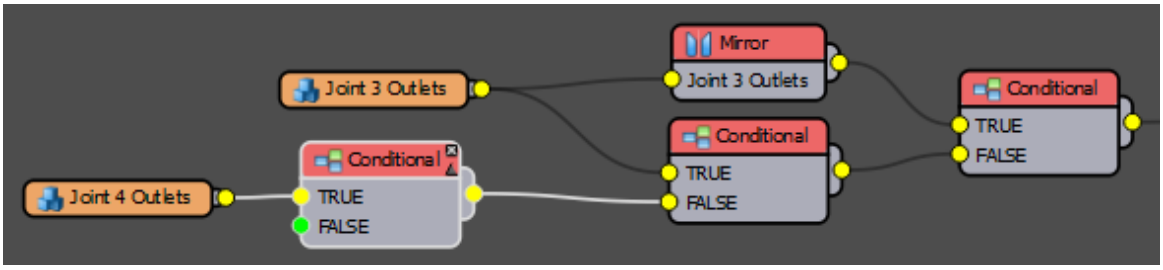
5. Select the new Conditional operator and turn on **Properties** > **Vertex** > **Type**. Select **Corner** from the dropdown menu.



6. You know have an intersection created by smooth vertices and a branch right on corner vertices. To finish up we'll create a third **Conditional** operator to create a branch left segment. Wire the second Conditional operator to the third Conditional operator's **False** input.
7. Select the third Conditional operator and turn on **Properties** > **Vertex** > **Type**. Select **Smooth** from the dropdown menu.

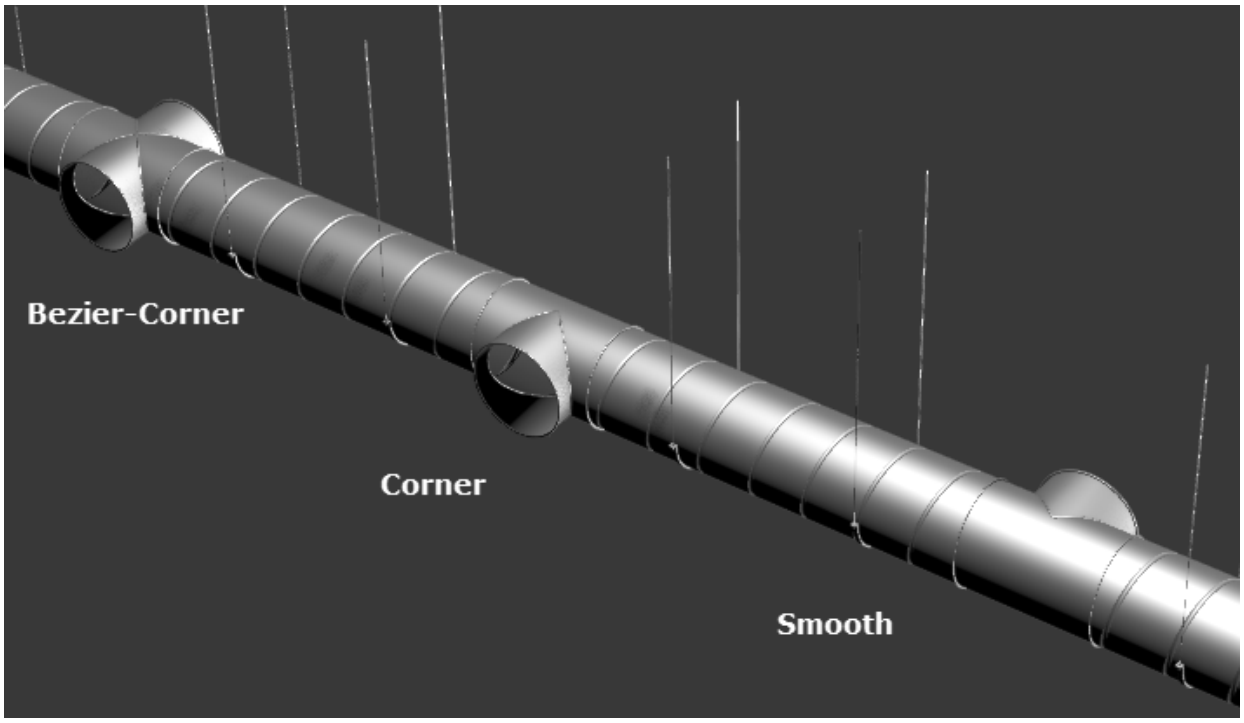


8. Create a new **Mirror** operator and wire the *Joint 3 Outlets* segment to its input. Go to the Mirror node's **Properties** and change the **Axis** to **Y**.
9. Wire the Mirror node to the third Conditional node's **True** input.



10. Wire the third **Conditional** node to the generator's **Corner** input.

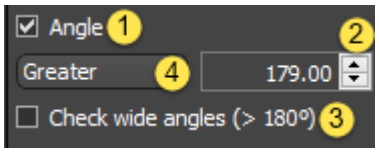
You should now have 3 different corner types shown along the length of the spline. Experiment with changing the vertex types on the spline to see the 3 Conditional operators in action.



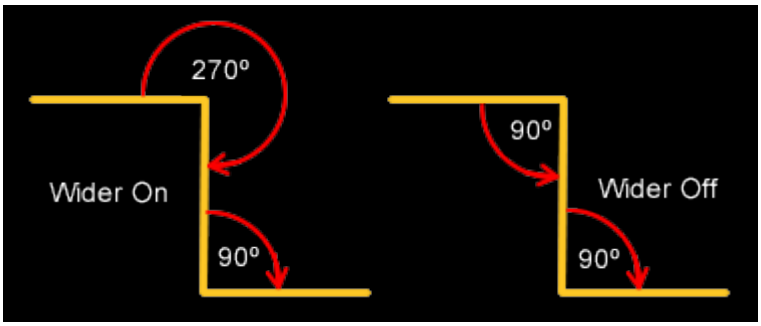
The Vertex Type condition is usually used in conjunction a generator's Corner input to create different types of corners, as we saw in the examples above. However it can also be used with the default input in which case the conditional node tests for the vertex at the **beginning** of the current spline section.

Exercise: Creating conditions based on the angle of the corner

As well as the type of vertex, the conditional operator can also evaluate whether the current corner is greater or less than a specified value. To use this feature use the following procedure.



1. Turn on **Properties** > **Vertex** > **Angle**
2. Enter a value for the **Angle** parameter.
3. Choose whether the angle should be evaluated between 0 - 180 degrees or **Wide** where the full 0 - 360 degree range is used.

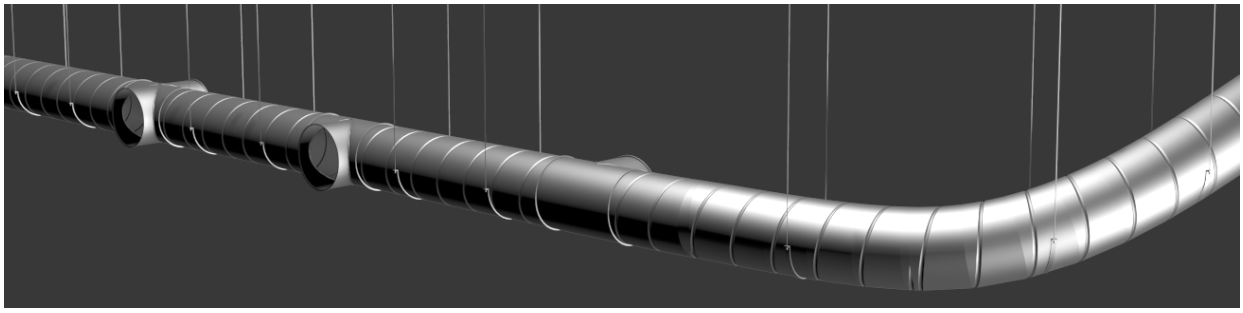


4. Choose whether you want to return True if the current corner is **Greater** or **Less** than the Angle value.

In our ducting example we'll automatically remove the corner segments created in the previous exercises unless the corner in close to 180 degrees. This will allow for bevelled corners in addition to preformed junctions.

1. Continue with *rc_vertex* and open the style editor.
2. Add a fourth **Conditional** operator between the existing Conditional nodes and the Generator's Corner input.
3. Select the new conditional node and turn on **Properties** > **Vertex** > **Angle.**
4. Enter a value of 179 degrees in the Angle field.
5. Ensure **Check Wide Angles** is **off.**

You can now create bevelled and curved sections as well as adding 3 types of junction. Where a corner is less than 180 degrees, the junction segments are automatically disabled.

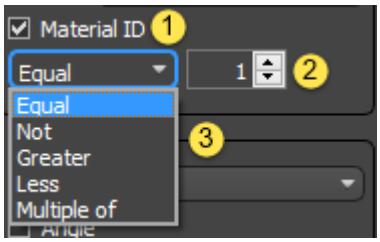


9.3 Segment based conditions

In the final section on the Conditional operator we'll look at how it can be used to test attributes of the array's segments.

Exercise: Creating conditions based on the segment count.

The conditional operator can use the number of the current segment in the array measured against a fixed value to return one of two segments. It works by counting the number of segments on the X or Y axis and then returning true or false dependent on if the number is Equal, Not equal, Greater than, Less than, or a Multiple of a set value. To use this mode:



1. Turn on **Properties** > **Segment** > **X Counter** or **Y Counter**
2. Enter a Count value
3. Choose whether you want to return True if the current segment has an X or Y count that is equal, not equal, greater than, less than, or a multiple of the count value.

> Using the Multiple-Of option, This operator can be used to create segments at certain intervals. In our air conditioning example we will exploit this feature to change the spacing of the side vents along the ducting. To do this:

1. Select *rc_segment_count* and open the Style Editor.
2. Wire the the existing **Compose** operator to the **True** input of a new **Conditional** operator. Wire the Conditional node to the **Generator's Default** input.

3. Select the Conditional node and turn on **Properties** > **Segment** > **X Counter**
4. Set the mode to **Multiple Of** and set the value to **5**. You will now get a vent in every other bay.
5. Try experimenting with different values. You'll notice that only odd numbers work, this is because the vents and the hangers are alternating, so the hangers land on even numbers while the vents are on uneven numbers.

In this chapter each of these conditional tests has been created in a separate style for clarity, but of course there's no reason why they can't all be combined. In *chapter_9_ducting_end.max* you will find a style which combines many of the exercises in this chapter. This style uses the following conditions:

- **Vertex Type**. Choose from 3 types of junction by using Smooth, Corner, and Bezier - Corner
- **Material ID.** Choose to turn side vents on/off using Material ID 1 or 2
- **Spline Segment Type.** Use line segments for regular ducting and Curves for articulated ducting.
- **Segment Counter.** Used to change the spacing between side vents
- **Selector Operator.** Turn on and off the end grills.

Feel free to experiment and adapt this style to learn how many conditional nodes can work together to create a highly adaptive style.

9.4 Using the Selector Operator to create conditional relationships

For more advanced usage, the selector operator can also be used to create conditional relationships. There are two ways of doing this. You can either use the operator's built in mode to assign segments based on the material IDs as demonstrated earlier in this chapter. Or you can export the Index value and use this in conjunction with the Arithmetic node to create more complex conditional statements based on built in variables.

Both these techniques enables you to create conditional relationships with fewer nodes than is possible with the Conditional operator, but using the arithmetic node also gives you access to some variables that are not available elsewhere. You can see a full list in Chapter 3 - Wiring and Attributes, but to explain how to use Expressions with the Selector operator we'll focus on just one - the Position on Y Spline attribute which cannot be accessed from the Conditional operator.

Exercise: Creating conditions using the Selector operator and expressions.

The Arithmetic node is capable of creating conditional expressions using **If** statements. The syntax of an if statement is fairly straightforward.

if(p , q , r)

Where **p** is the condition that is being tested, **q** is the value that is output if the test is true, and **r** is the value that is output if the test returns false. It is also possible to nest If statements using this format.

if(p , q , if(p , q , r))

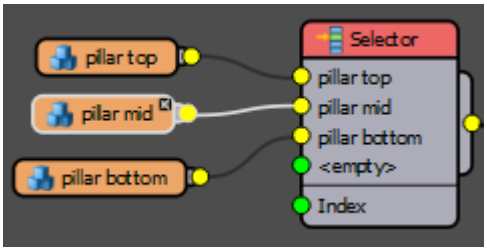
In this example the nested If statement has been underlined. In practice you may find it is much clearer to format the expressions so that the if statements are on separate lines, especially if you have a lot of them:

if(p , q ,
if(p , q ,
if(p , q ,
if(p , q ,
r))))

In this exercise we'll use this technique to add the top and bottom to a pillar which will have an adjustable height. This style is made from 3 segments in a two dimensional array. The pillars will be added to the Evenly input so that we can easily control their spacing. To create this, follow these steps.

1. Open chapter_9_pillar_start from the downloads for this guide.

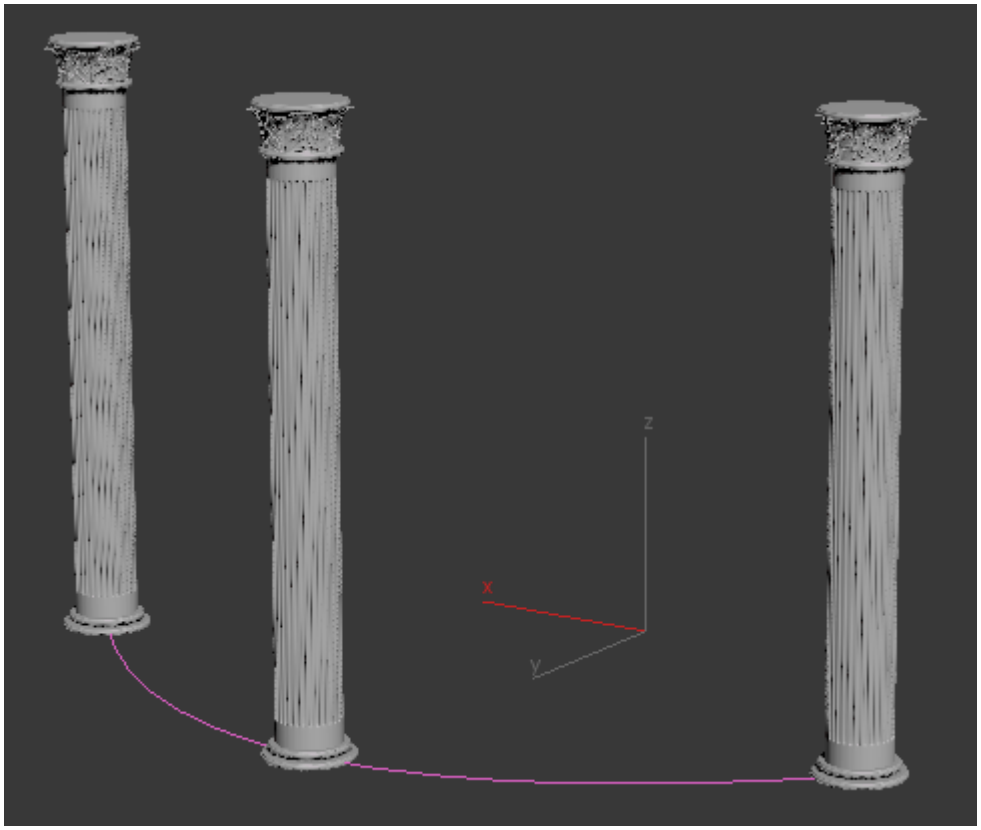
2. Select the RailClone object and open the style editor. All the necessary segments have already been added to the style.
3. Create a new **Selector** operator.
4. Next we'll wire the segments to the Selector operator's inputs. The order is important - the index numbering is built from the top of the list with an origin of 1. Wire the *pillar top* to input 1, *pillar mid* to input 2, and *pillar bottom* to input 3.
5. Wire the **Selector** node to the generator's **Left Side**, **Right Side**, **X Evenly** and **Inner Corner** inputs. With the exception of the sides, none of these inputs have a way to target the top and bottom of the column so we'll build an expression instead.
6. Right click on the Selector node and **Export the Index**.



7. Create a new **Arithmetic** node and wire it to the **Index** socket.
8. Select the Arithmetic node and change the mode to **Expression**.
9. Open the Expressions editor and enter the following:

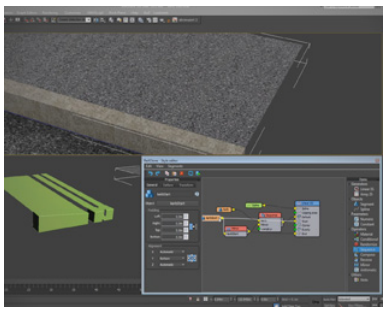
```
if( YSplinePosition=0 , 3 ,
if( YSplinePosition=1 , 1 ,
2 ))
```

As you can see, this is a nested If statement that is testing a segment's position on the Y axis. The position is measured from 0.0 at the the start of the array to 1.0 at the end. The first if statement is testing to see if the segment is at the start of the spline (YSplinePosition=0.0), if it returns true then the expression outputs the value 3 to the Selector operator which picks the pillar bottom. If it is false then it tries the second If expression which is checking to see if the segment is at the end of the spline (YSplinePosition=1.0), if this returns true then the expression outputs the value 1 to the Selector operator which picks the pillar top. If this return false then it outputs a value of 2 which tells the selector operator to pick the pillar middle.



By using the if statement in the arithmetic operator in this way you are able to access many hidden variables in RailClone and create more sophisticated conditional relationships between parts without having to create a lot of additional nodes.

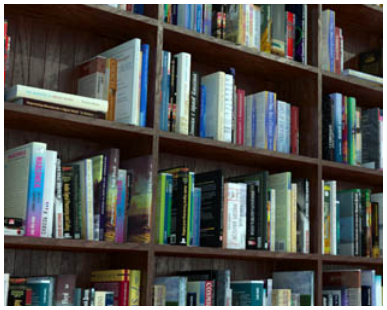
9.5 Related Tutorials



create_a_drop_kerb

Creating a Drop Kerb

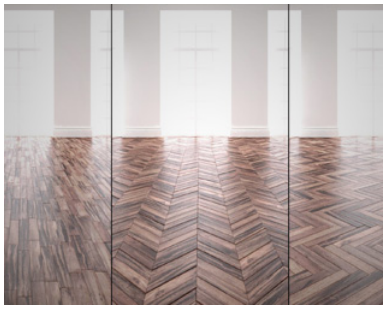
In this Tips & Tricks tutorial we look at how RailClone can be used to place drop kerbs using material IDs assigned to the base spline.



creating_randomised_books

Creating Randomised Books

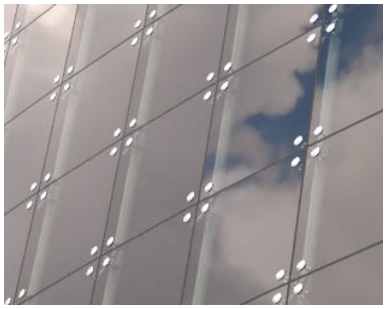
In this tutorial we explain how to use Conditional operators to test the position on a spline segment and prevent books from being created too close to the ends of a shelf where they might intersect with the vertical elements of a bookcase.



create_a_parquet_floor

Creating Parquet Floors

In this exercise we use the Conditional operator to create the pattern on the Strip floor. Using the conditional operator a half sized plank is added to the first segment in alternate rows to create an offset pattern on the floor.



creating_spider_glazing

Creating Spider Glazing

In the spider glazing tutorial we show how to use the conditional operator to target parts of an array that do not normally have an input. By testing for a segments position and count on the Y spline, it is possible to add segments at the start and end of a column.

Documentation

For more information please see the Operators, 1D arrays - Generator L1S, and 2D arrays - Generator A2S, sections of the online documentation.

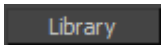
10 10 - Creating Presets

In this chapter we share tips and tricks for creating your own style presets for RailClone. If you regularly use the same styles in your scenes, it makes sense to add them to the built-in library browser for easy access, and If you work in a studio you can easily save library configuration files in a networked location so that artists can share and collaborate on a central repository. In the first half of this chapter we focus on the the essentials of using the library browser to create and edit libraries and in the second half we'll demonstrate advanced tips that show how to support multiple renderers and trigger maxscript events to extend the functionality of the library. This section is aimed at users that would like to share their libraries either in the same studio or even for distribution online. We'll explain how to package assets and make them more accessible for users with different configurations.

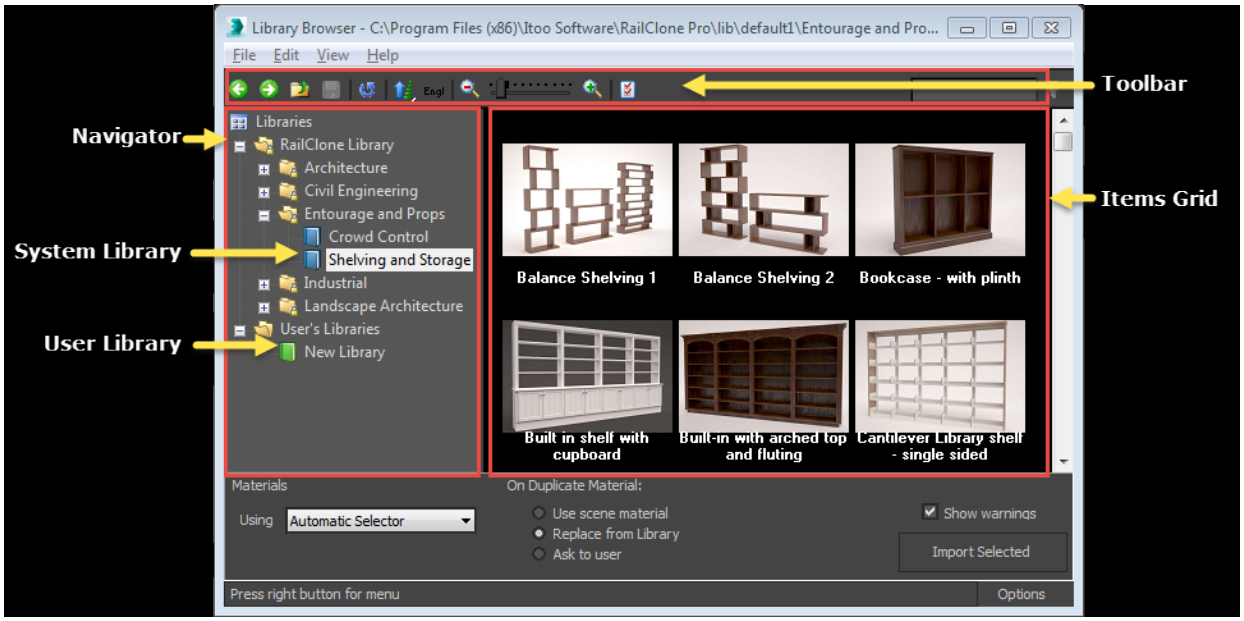
In the exercises below we'll create a library to store the Picture frame example used in Chapter 2, this is a simple example to illustrate the procedures but the same would apply for any RailClone object and libraries of any size or complexity.

10.1 Basic Library Usage

The library interface

If you're new to RailClone, before creating your own libraries it's worth taking some time to recap the library interface. To open the library click on 

in the **Style** rollout.



On the left of the interface is the navigator, use this to select, edit and open your libraries. You'll notice that there are two types of folder. System folders, which are represented in blue, denote the libraries that come packaged with RailClone. These are **read-only** and can't be modified from the library browser. Below these are the editable user folders represented by a green folder. This is where your custom libraries can be found.

After a fresh install of RailClone you'll have one location already added called **User's librarys**. The default location for this is *C:\Users\<username>\Documents\RailClone\libs* but you can save libraries on any local or network drive in any folder. We'll look at how to edit and add locations a little later in this tutorial.

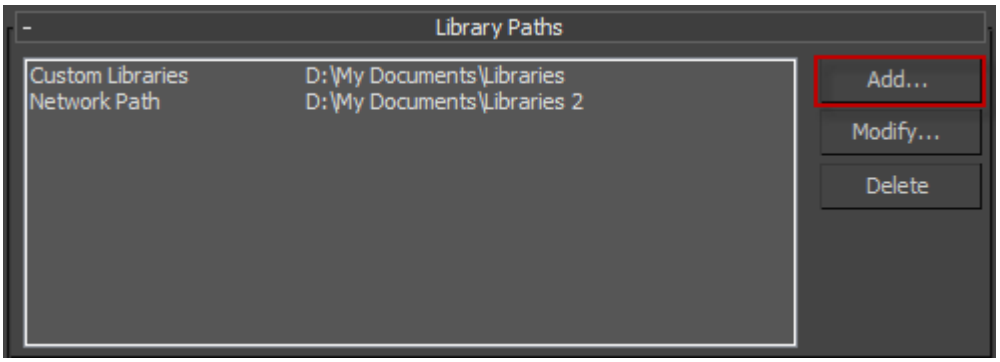
On the right is the **Items grid**. This view previews the contents of the selected library and allows you to select presets. Along the top is the toolbar, with several command buttons to navigate, save and refresh the library as well as changing the size of the thumbnail images.

There may be times when you'd like to make a copy of an existing library. To do this select a library and then dragging it to any user folder while holding down CTRL. All the necessary files are copied to the new location and you can now edit this version. You can also move files between folders just by clicking and dragging in the navigator view, there's no need to open Explorer for basic file operations!

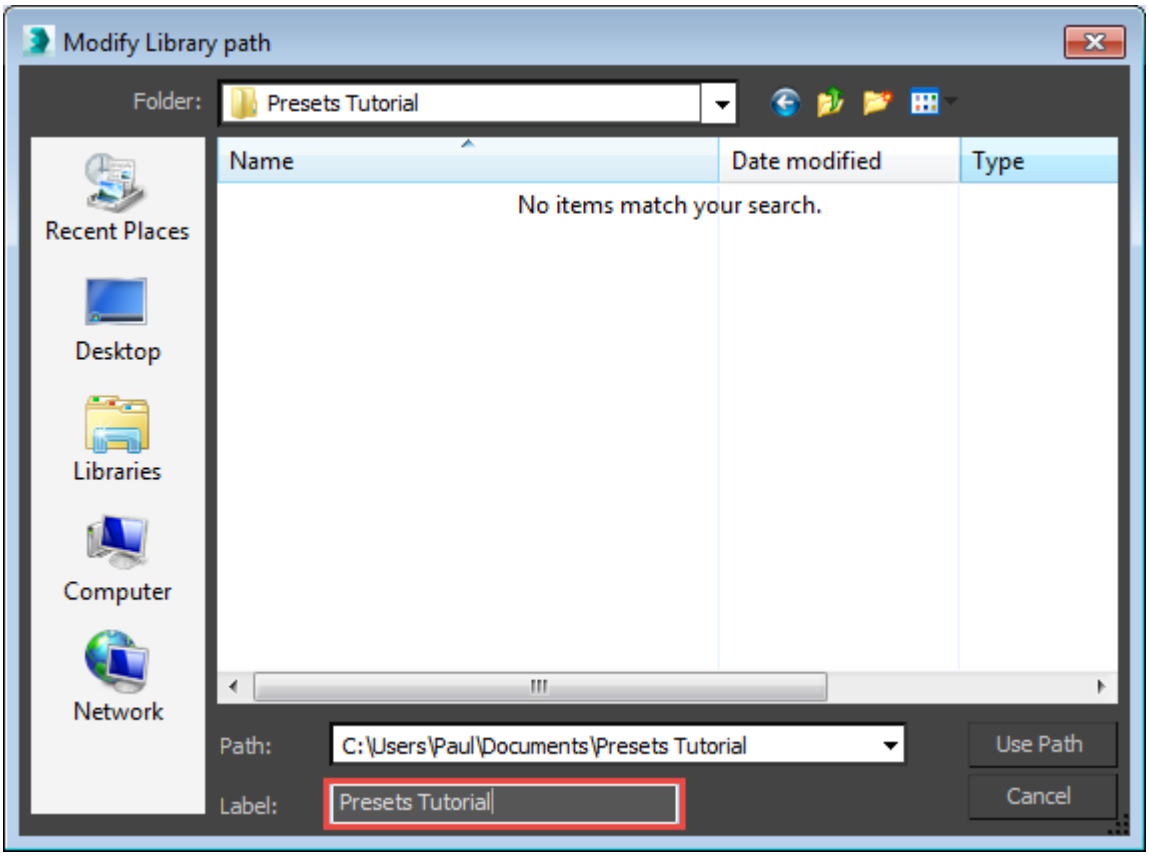
Exercise: Creating a new Library

The first thing you need is a root folder. This is the path that RailClone's library browser will search in for library configuration files. As mentioned above, there's already a User folder set up, but for the purposes of this exercise we'll add a new directory. In this example the path is created locally, but there's no reason why this can't be shared network location so that assets can be easily accessed by multiple artists in a studio environment. To add a new library location:

1. With the library editor still open go to **File** > **Preferences...** or click from the top toolbar.
2. The **Library Paths** rollout shows a list of locations in which RailClone will search for libraries. To add a new location click the **Add** button.

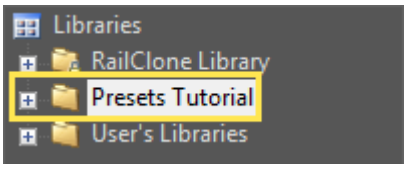


3. In the **Add New Library Path** window, navigate to a folder in which you wish to save your libraries. If necessary click to create a new folder.
4. In the **Label** field enter the name to display in the tree navigator.



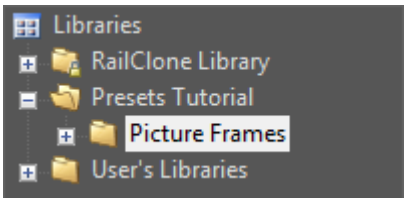
5. Click to add the location to the library and return to the preferences.

6. Click **OK** and you should now see the new location displayed in the tree navigator on the left hand side of the library browser.



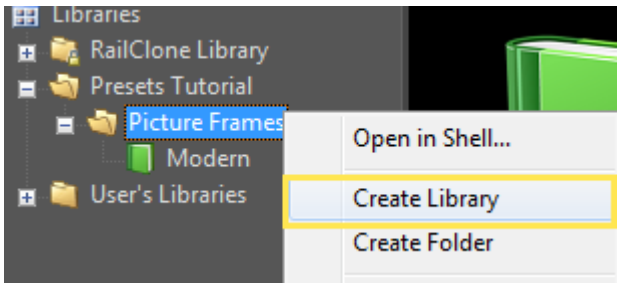
Now that we have a new folder created, we can add a library, but it's worth mentioning that it's also possible to add additional sub-directories to this folder directly within the Library Browser. To do this

1. In the tree navigator, right click on the new location name and select **Create Folder**.
2. A new path will be created and the name is highlighted for you to enter a name. If you are planning a large library this makes it easy to organise your content.



Finally let's add the library itself.

1. In the tree navigator, right-click on the folder where you would like to save your new library and select **Create Library.**

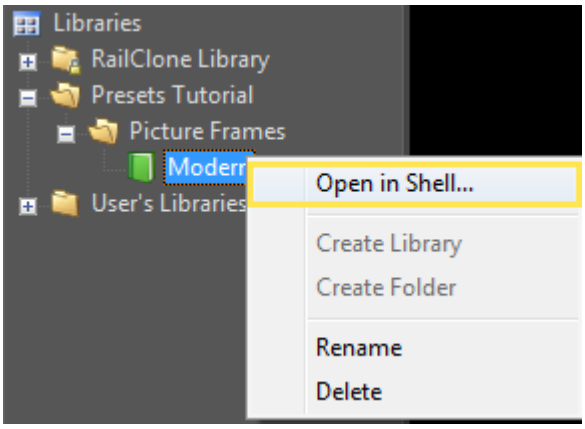


2. A new library is created and the name will be highlighted, enter a title and press return.

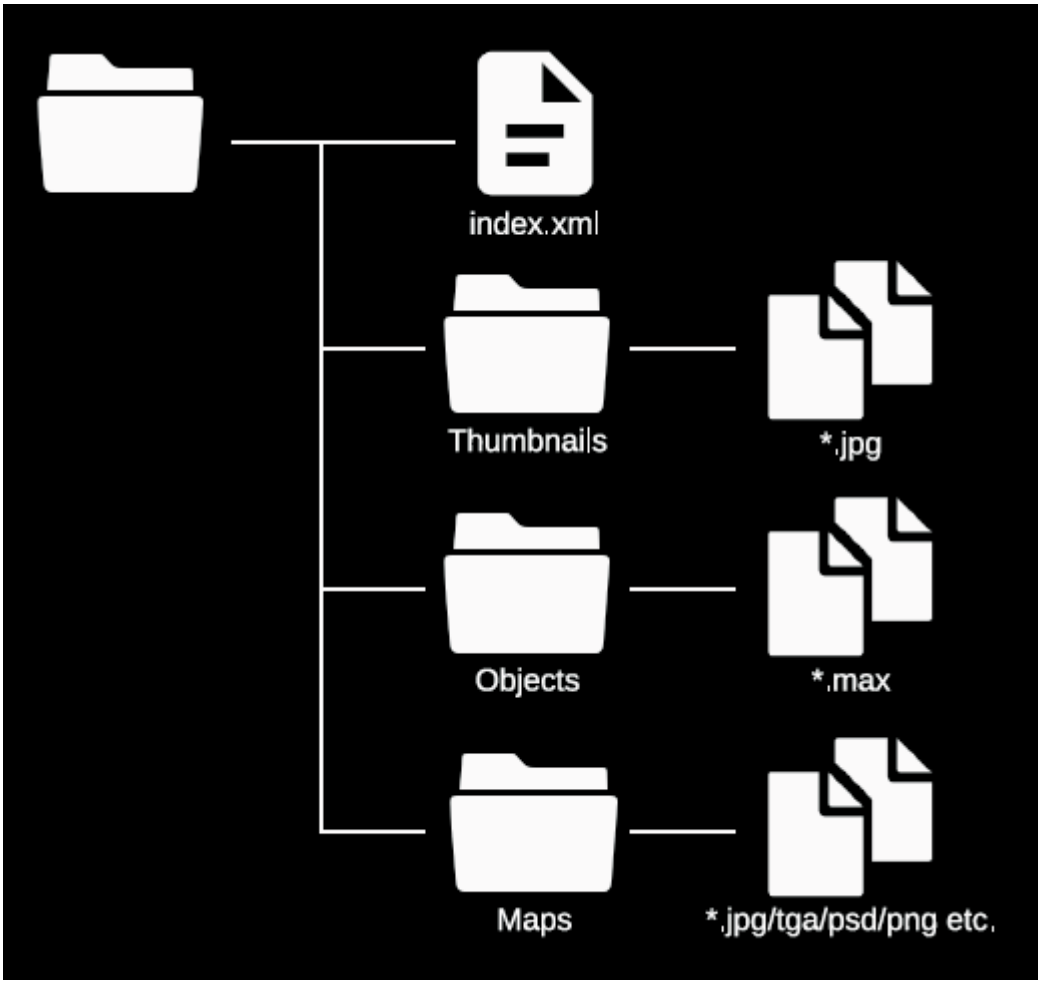
When you create a library, a new folder is added that includes an XML file and a sub-folder for thumbnails. This and any other folders found in the same location as an index.xml file are not displayed in the Library navigator. This allows you to organise your assets by adding additional directories to separate models, maps and materials.

Although you can easily create libraries entirely from the library browser interface, if you plan to share library files with colleagues, other users, or offer them for sale, it is helpful to understand how library files are stored behind the scenes. Fortunately it's not complicated, and the Library browser does most of the work for you. Let's look at how a library is organised:

1. To open the location of a library's files, right click on the name of the library and select **Open In Shell**.



2. Windows Explorer will open showing the contents of the library's folder. In here you'll see an index.xml file that contains all the libraries settings, and a folder named thumbnails that is used for the the preview images show in the items view.
3. To organise things further create a new folder for the .max files, and another for textures . You can use any names, but we recommend adding a folder called ***objects*** for the .max files and ***maps*** for your textures. The diagram below illustrates a typical library setup.



The XML file itself contains the configuration for the current library, including the location and name of the .max files and objects referenced, the materials to use, which thumbnail to display and much more. As we'll see later, it's also possible to trigger maxscript events from the XML file to add additional functionality to the library such as detecting the current renderer, adding new user paths and much more.

> Never save your own libraries in the same location as RailClone's built in presets. This directory will be wiped whenever you update the software and you'll lose anything saved there.

To create A "Picture Frames" library we need to add the files to the folders we just created. To do this:

1. Add the preview renders to the /**thumbnails** folder. A thumbnails should be named after its corresponding object in the .max scene. If the names match, the library will automatically load the thumbnail. If the names are different you will have to add the thumbnails manually.

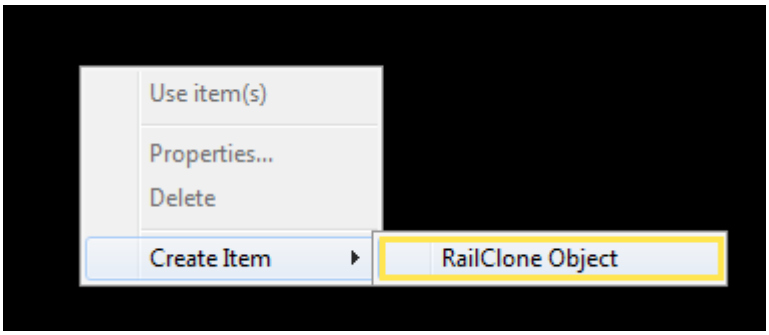
2. Add the *.max* files to the **/objects** folder
3. Add the textures to the **/maps** folder.

Exercise: Adding a new preset to the library

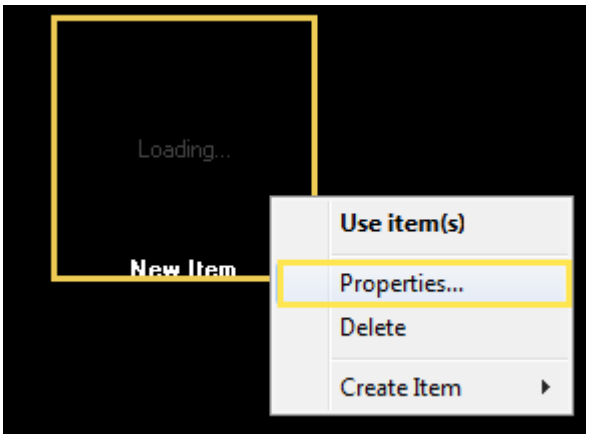
Now that the source files have been moved to correct locations we're ready to add some objects. To start with let's look at how to add a single item:

1. Open the library browser by clicking

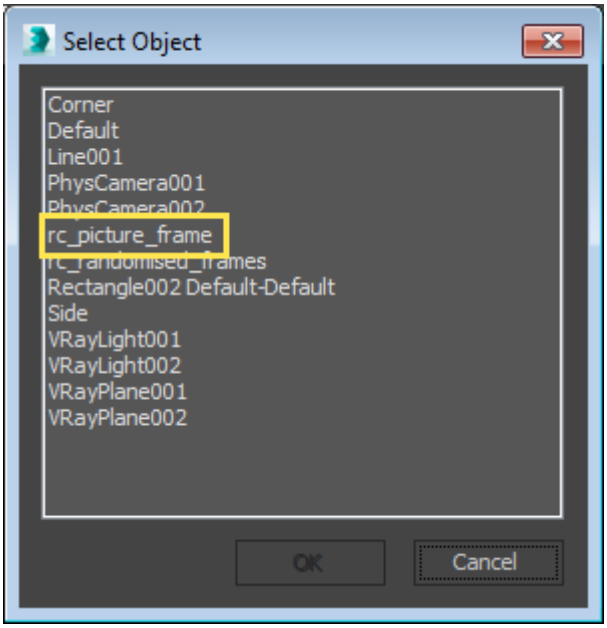
2. Select the new library from the Navigator and right-click in the items view. Click **Create Item** > **RailClone Object**.



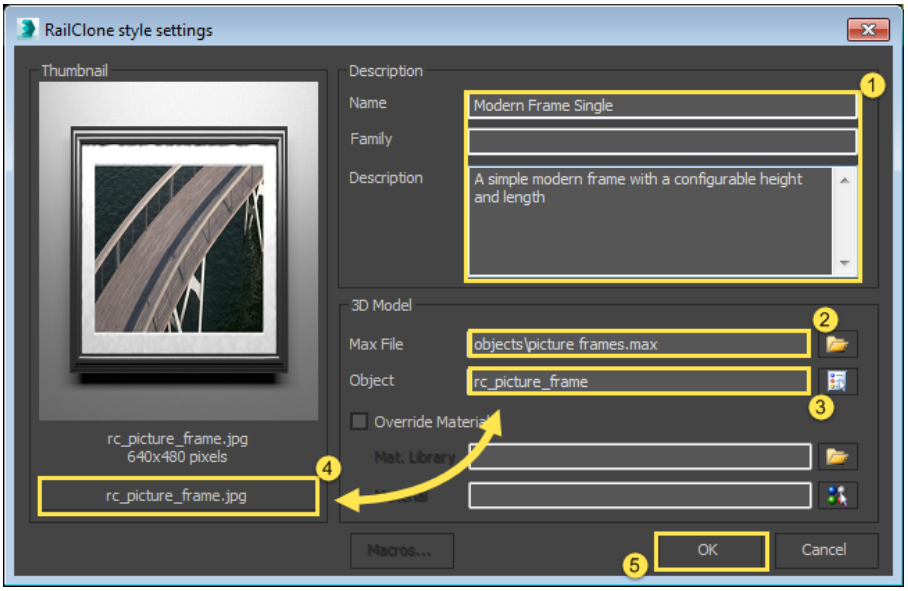
3. This creates a new empty item. Right-click on the item and select **Properties...**



4. Fill in the **Name** values. Other values like **Family** and **Description** can be left blank if not required
5. Click on the folder item next to the **Max File** field and select a .max file. In this example we'll select the *picture_frames.max* scene file.
6. A list will open containing the objects in the scene, pick the item you wish to add to the library, in this case we'll add **rc_picture_frame**



7. You'll notice that the thumbnail appears automatically. This is because there is an image in the thumbnails folder that shares the name buttercup_001 - the name of the object we just selected. If you prefer to select the thumbnail manually, just click on the Thumbnail Name button below the preview window.
8. Click **OK**.



9. Click from the top toolbar to save the changes to the library.

Once you've added an object to the library you can open the XML to see how RailClone saves this information. Even if you're not familiar with XML syntax, this mark-up is fairly self-explanatory:

```
<?xml version="1.0" encoding="utf-8"?>
<library version="100" order="1" masterScale="0.393700787402" xref
="1" bitmap="">
    <item class="maxobject">
        <name lang="english">Modern Frame Single</name>
        <name lang="latin">New Item</name>
        <family/>
        <description>A simple modern frame with a configurable
height and length</description>
        <bitmap>rc_picture_frame.jpg</bitmap>
        <maxfile>objects\picture frames.max</maxfile>
        <object>rc_picture_frame</object>
    </item>
</library>
```

The details for each item are stored between <item></item> tags with the class "Max Object". Inside this there are tags for the fields that we just entered in the library browser. If you look at the <maxfile> tag you can see that paths are relative to the index.xml file.

If you prefer to create libraries purely by editing the XML, it's a simple case of duplicating the <item></item> tags plus their contents, and changing the values.

Managing Materials

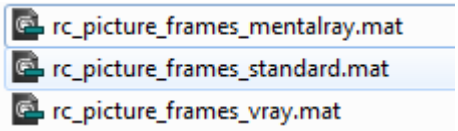
Unless you specify otherwise, the materials applied to the original source objects will be imported along with the geometry, so if your only supporting a single renderer or you don't plan to distribute the library there's probably no need to go through the following steps and **your library is done!**

10.2 Advanced Library Usage

If however you would like to add the ability to select materials to support multiple renderers, here how this works.

You may have noticed our built in libraries have the option to select a different material library for each supported renderer using a drop down list. This is achieved using a macro that inserts a different name into a material library's path depending on the option selected in the material selector dropdown. For this to work you need **3** ingredients:

- A material Library for each renderer you wish to support. We recommend saving them in a new **matlib** folder in the library's directory. The file names should be structured so that they share a **common** section that is the same for every file, plus a **variable suffix**. In the examples shown below the common portion is **_rc_picture_frames_** and the variable portion is **standard**, **mental**, or **vray**



The names of the materials in the material library must be identical to those applied to the source objects. The library will search the library for a material of the same name as the one currently applied to the object.

- A code snippet added to the XML to create a macro that can be used to tell the library browser which matlib to use. Note that the contents of the value attribute match the variable suffix used in the matlib's file names.

```
<select class="material" setmacro="RAILCLONE_RENDERER"><option value="mentalray"
>MentalRay</option><option value="vray">VRay</option><option value="standard">Standard<
/option></select>
```

- For each item in the library, a material override with a path to a matlib that uses the macro value in its file name. You can add this either in the library browser or in the XML. The name is based on the material library's folder name + it's filename + the name of the macro, for this example is should read *matlibs\rc_picture_frames_$(RAILCLONE_RENDERER).mat*

Using these 3 ingredients, we'll demonstrate how to add the ability to select materials to our picture frame library for Mental Ray, Corona and V-Ray:

1. First, in the library root, create a new folder called **matlibs** and copy the 3 provided material libraries. Note that each file's name is identical except for a renderer suffix. This value is arbitrary, but we recommend you maintain consistency by using the naming conventions shown in the table below. The first column shows the value returned by the macro and the second column is the renderer to which it relates.

Value	Renderer
standard	Scanline
vray	V-Ray and V-Ray RT
corona	Corona
mentalray	Mental Ray and IRay
octane	Octane

Value	Renderer
thea	Thea

Using these we create material libraries with identical names **except** for a prefix or suffix that contains the renderer value. For example in this tutorial the matlibs are named *rc_picture_frames_**standard**.mat*, *rc_picture_frames_**mentalray**.mat, and rc_picture_frames_**vray.**mat.* You'll find these in the the *Matlibs* folder in the downloads for this guide.

1. Now we need to create a macro that returns the values shown in the table above depending on the selection in the material dropdown menu. To do this open the library's *index.xml* file in your favourite text editor and underneath the opening **<library ...>** tag add:

   <select class="material" setmacro="RAILCLONE_RENDERER">

   The important part of this is the **setmacro** attribute which is used to name the macro. To call this macro from the library browser you just use this name in the format **$(RAILCLONE_RENDERER)**.

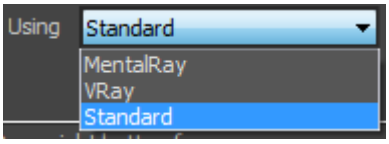
2. Next we add the renderer choices using the **<option></option>** tag.

   <option value="mentalray">MentalRay</option>
   <option value="vray">VRay</option>
   <option value="standard">Standard</option>

   In each case the text between the opening and closing tags displays in the drop down list and the value is the text that the macro returns when called.

3. Finally, add a closing **</select>** tag.

4. Save the file and open the library browser. You should now see the 3 options in the Material drop down menu.

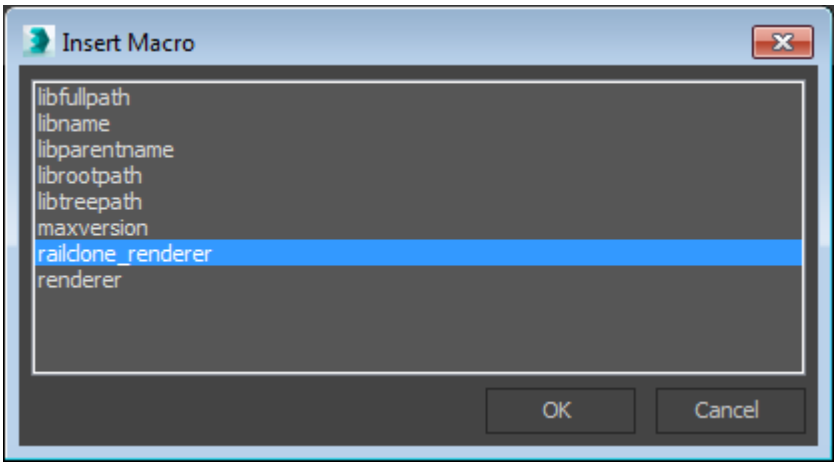


Now that the options are working we must override the materials currently applied to the object with those from the material library, To do this:

1. Open the Library Browser. Right click on a preset and open the **Properties**.

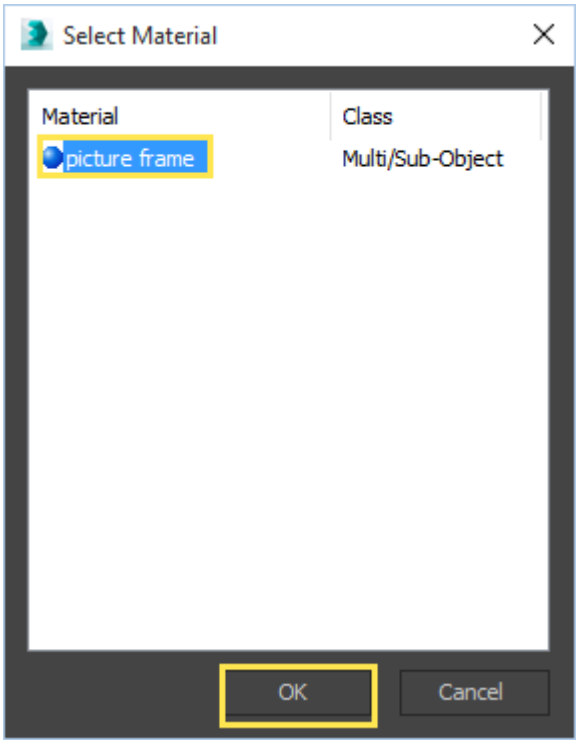
2. Activate **Override Material** and click on to select one of the material libraries in the matlibs folder

3. In the Mat. Library field we want to replace the renderer suffix with the macro we just created. Select the renderer part of the file name (matlibs\rc_picture_frames_vray.mat) and either type the macro name manually or click on Macros... and a list will open. Double click on ***RAILCLONE_RENDERER*** to insert the macro into the material library path.



4. The path should read **matlibs\rc_picture_frames_$(RAILCLONE_RENDERER).mat**

5. Next select the **Material** to use from the library. Click on to open the material Picker. Select a material and Click OK.



6. To test the file, set the material selector to Mental Ray and open an object from the library, the material should use Arch and Design shaders. Switch the renderer to Standard and reimport the same object. Standard materials should now be used on the object, replacing the Mental Ray versions.

7. If you have multiple objects in the library you may find it is faster to do this in the *index.xml* file by adding or copying the <matlib></matlib> tag for each item.

```
<item class="maxobject">
    <name lang="english">Modern Frame Single</name>
    <name lang="latin">New Item</name>
    <family/>
    <description>A simple modern frame with a configurable height and length</description>
    <bitmap>rc_picture_frame.jpg</bitmap>
    <maxfile>objects\picture frames.max</maxfile>
    <object>rc_picture_frame</object>
    <matlib>matlibs\rc_picture_frames_$(RAILCLONE_RENDERER).mat</matlib>
</item>
<item class="maxobject">
    <name lang="english">Modern Frame Single</name>
    <name lang="latin">New Item</name>
    <family/>
    <description>A simple modern frame with a configurable height and length</description>
    <bitmap>rc_picture_frame.jpg</bitmap>
    <maxfile>objects\picture frames.max</maxfile>
    <object>rc_picture_frame1</object>
    <matlib>matlibs\rc_picture_frames_$(RAILCLONE_RENDERER).mat</matlib>
</item>
```

In this example we are replacing the material on the object with another of the same name from a material library. It is also possible to assign a completely different material from a material library. To do this in the **Item Settings** click on and pick a new material from the library. This will add a new line to the XML, for example:

<material>_rc_presets_tutorial</material>

Using Maxscript to detect the current renderer

For a more advanced setup it is also possible to extend the library using maxscript functions. For example, you may want to add the ability to automatically detect the active renderer instead of selecting it manually. To set this up:

1. Create a new Maxscript (.ms) file in the **same directory** as the index.xml file. For this tutorial we have included a maxscript file called *tutorialLibraryFunctions.ms* which you are free to re-use . This tutorial isn't intended to teach maxscript, but even so the function we need is easy to understand:

```
fn getRenderer = (
    local renderClass = (classof renderers.current);
    if ((renderClass == VRay) or (renderClass == VRayRT))
then "vray"
```

```
    else if ((renderClass == mental_ray_renderer) or
(renderClass == iray_Renderer)) then "mentalray"
    else if (renderClass == Octane3dsmax) then "octane"
    else if (renderClass == Thea_Render) then "thea"
    else if (renderClass == CoronaRenderer) then "corona"
    else "standard"
)
```

This creates a new function called getRender. It uses *classof renderers.current* to return the active renderer and assign it to a variable called *renderClass*. The function then uses a series of if-else conditional statements to test *renderClass* against a list of renderers and return a string value if it tests true. You should notice the renderer name being returned is the same as shown in our table earlier and also the same values as used already in the drop down list and in the matlib file names. If you are not supporting a particular renderer it can be removed from the function and likewise additional renderers can easily be added using the same format.

2. Next we need add a new automatic option to the dropdown material selector. To do this open the *index.xml* file and add a new option inside the existing <select></select> tags. For the automatic selector to appear at the top of the list, make sure it is the first option. Add the following:

   <option value="@tutorialLibraryFunctions.ms:getRenderer()">Automatic Selector</option>

   When this option is selected, instead of the value being a string, it is injecting a macro that will call a maxscript file and function. Note that the name of the maxscript file is preceded by an **@** symbol and it is separated from the name of the function by a colon (**:**).

3. To test this works, change the renderer to Mental Ray and import an object from the picture frames library using the new materials > automatic setting. If you drag the material from the geometry rollout to the Material Editor you should see that the material uses Arch and Design materials. Then switch the renderer to V-Ray and import the same object - you should now have V-Ray Materials!

Location of the .ms file

When a maxscript file is called by the library browser it will search through all valid folders within the library path. We recommend saving Maxscript files either in the same location as index.xml or if you plan to re-use the same functions for multiple libraries in the root directory. (these are the directories shown in the **File** > **Preferences** > **Library Paths** list.

Managing Maps

The remapping of bitmap paths is not handled automatically by the Library Browser. To ensure that maps are located correctly you will need to add the maps directory to the users paths manually. To do this:

1. In 3ds Max, go to **Customize->Configure User Paths->External Files**
2. Click Add... and locate the maps folder then click Use Path to add the location.
3. The directory will now be used to search for missing map files. In our own libraries we strip the paths before saving the material.

If you have a large number of libraries it may be simpler to simply copy all of the maps to single central folder and add this to the user paths instead of adding a maps folder for each library.

In the next section we'll look at how this process can be automated using maxscript.

10.3 Running Maxscript functions

To add additional functionality to the library browser we have included the ability to run a maxscript function when either the library is opened or an object is imported. To do this you need two things. Firstly an .ms file containing the functions, as before this should be saved in the same location as index.xml or the root directory if you'd like multiple libraries to access the file. Secondly you should add a line to the XML to tell it which *.ms* file to load plus the function's name. The basic format for this is

<maxscript file="*filename.ms*" script='*functionName*();' execute_on="item_select"/> -- (Execute_on can also = "lib_open")

Where *filename.ms* is the name of the maxscript file, *functionName()* is the name of the function saved in the maxscript file, and *lib_open* or *item_select* is when the script should be executed.

To demonstrate this, we will add a code snippet that adds the maps folder path for the current library to the User Paths automatically. This is useful if you are distributing libraries as it will prevent users from having to add these paths manually. This function can be found in the *tutorialLibraryFunctions.ms* file we used earlier and is included in this download. The function is called addMapDirectory and is as follows:

```
fn addMapDirectory LibraryMapPath =
(
 local pathExists = false
 for i = 1 to mapPaths.count() do
 (
  if mapPaths.get(i) == LibraryMapPath then pathExists = true
 )
```

```
 if (pathExists == false) then
     (
         if queryBox "This library's maps folder should be added
to Customize->Configure User Paths->External Files.\n\nClick YES
to add it automatically\nClick NO if you prefer to configure the
map path manually." title:"Add Maps Path?" beep:true then (
         mapPaths.add LibraryMapPath
        )
     )
     LibraryMapPath
)
```

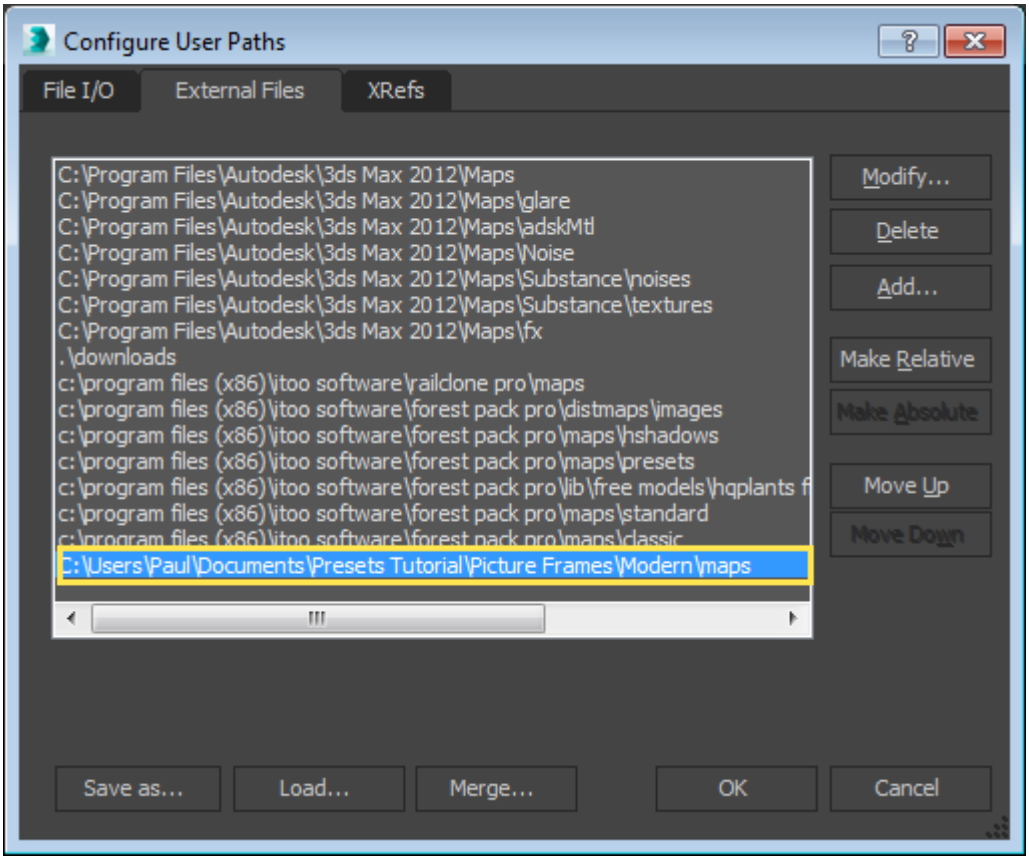
Without going into detail, this checks to see if the directory contained in the variable *LibraryMapPath* is already present in the User Paths. If it is missing a dialog will appear asking the user if they'd like to add it automatically. Let's add this to the library:

1. Open *index.xml*, and add the following line after the first <library> tag

   <maxscript file="tutorialLibraryFunctions.ms**" script='addMapDirectory("$(LIBFULLPATH)\maps");' execute_on="item_select"/>**

2. This is pretty self explanatory, except perhaps for "addMapDirectory("$(LIBFULLPATH)\maps");". In this case *$(LIBFULLPATH)* is calling a RailClone global macro that returns the full path of the current library. There are a few of these that you can use including libfullpath, libname. libparentname. and librootpath.

3. Save the XML file and go back to max to test the library. When you load in an object now you should get a dialog asking if you want to add the maps path. Click yes and then to test it has worked open **Customize** > **Configure User Paths** > **External Files**. You should see the maps path added as the final entry



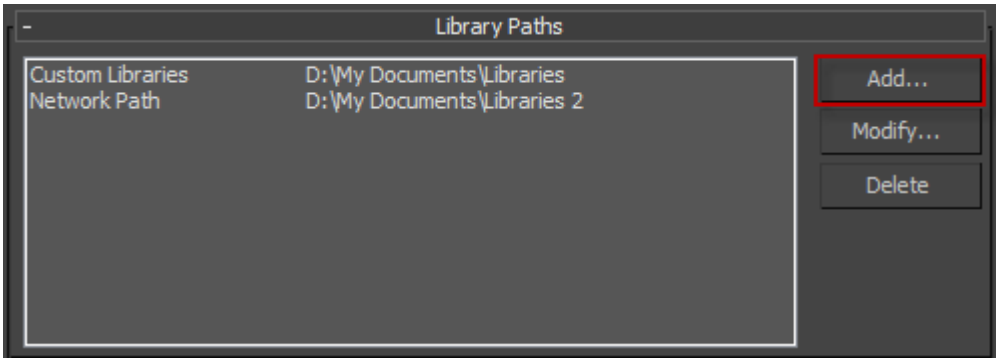
This is just one example, any maxscript can be triggered by the library including multiple functions.

That completes this library. If you want to distribute it, just .zip the folder and it's ready to go.

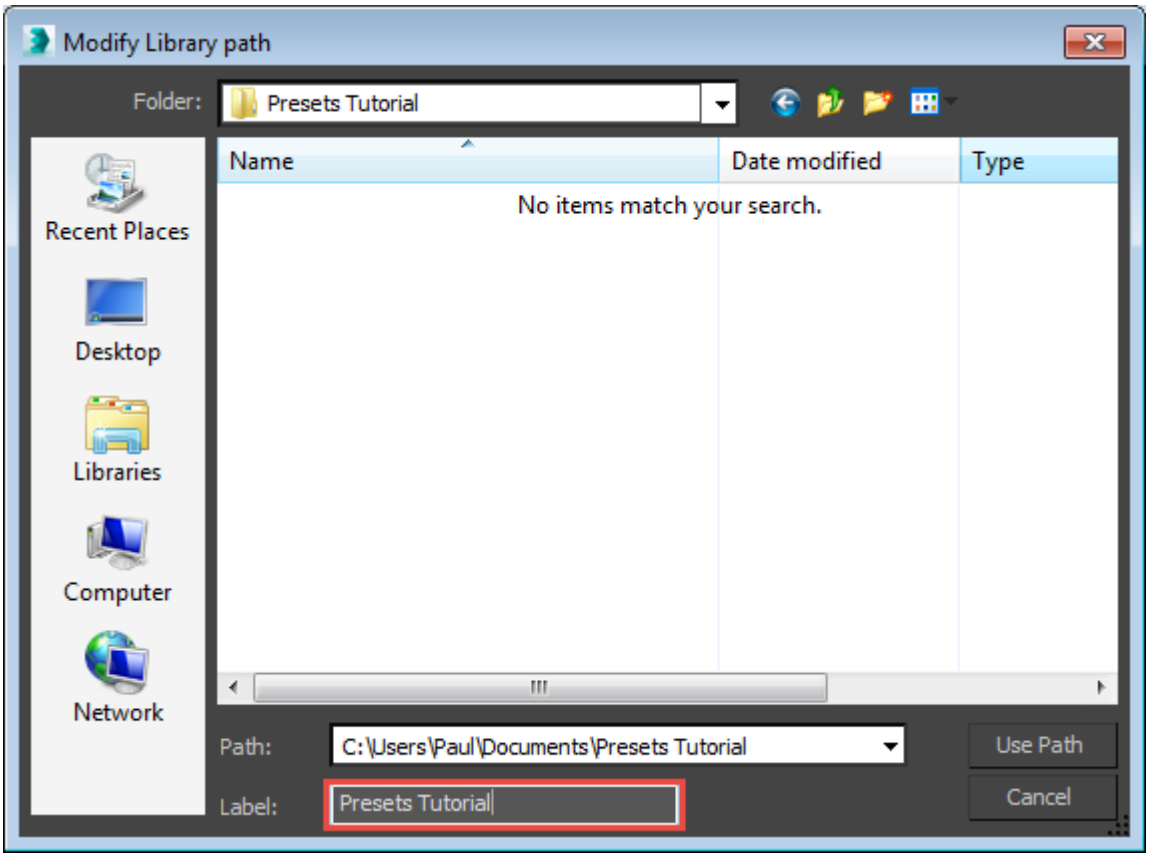
10.4 Installing Libraries

Now that this library is complete here's how to install it on another machine. This is very straightforward as it is only necessary for the user to add the library's path to their browser preferences. In fact it is the same technique as we used at the beginning of this tutorial to add a new folder location. If you're sharing your library, these are the kind of instructions you could distribute with your library files:

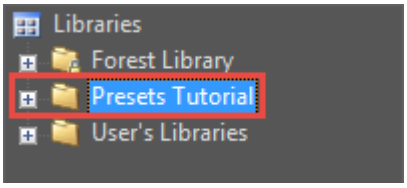
1. Unzip the library (if it's archived) and note the installation path, make sure that the original directory structure is retained.
2. Open RailClone's library browser and go to **File** > **Preferences...** or click from the top toolbar.
3. The **Library Paths** rollout shows a list of locations RailClone will search in for valid libraries. We'll add a new library so click on the **Add** button.



4. In the **Add New Library Path** window, navigate to the location of the downloaded or networked library.
5. In the **Label** field enter the name to display in the navigator.

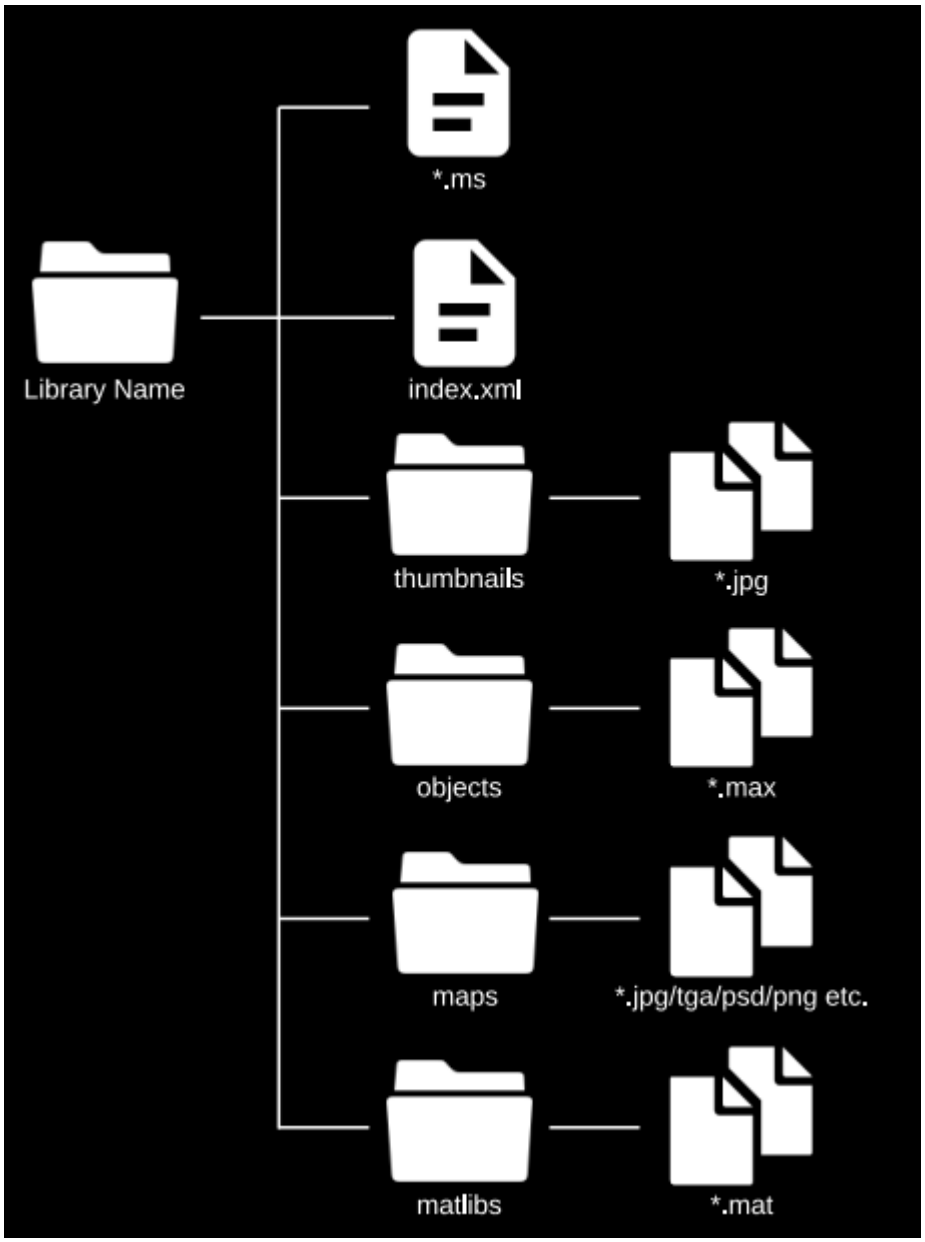


6. Click to add the location to the library.
7. Click **OK** and you should now see the new location displayed in the tree navigator on the left hand side of the library browser.



Conclusion

The final library created during this tutorial follows the good practice guidelines we recommend for well formed libraries that can be easily redistributed. In addition the thumbnails folder and index.xml file, the final structure includes a maxscript file for any additional functions and separate folders for objects, maps, and material libraries:



We've also looked at adding support for multiple renderers and how to add the libraries maps folder automatically to the Users Paths. Though we've only focused on a single library in this instance, multiple libraries can be included in the same location to make it easy to organise your content into logical categories.

11 11 - Next steps

Congratulations! You've reached the end of **Next Steps with RailClone**. If you followed this guide and completed the exercises then you've covered nearly all of RailClone's features. You should now have a good understanding of the principles that underpin RailClone's approach to procedural modelling and be able to use them in production.

To continue to get the most out of RailClone, we have a number of additional resources to help you develop your skills further.

11.1 Tutorials

We release regular tutorials that range from explanations of basic concepts, to advanced topics and demos of possible applications for RailClone. We've suggested relevant tutorials throughout the guide but to see the full list, check out the tutorials page. Come back regularly, we're adding new content all the time!

11.2 Forum

Join us in the Forum where you'll find a wealth of information and real-world uses for RailClone. We regularly respond to user's questions here and there is a huge amount of information that can be learned from these case studies. Many of the posts also include scene files that you can download to pick up new tricks.

11.3 Documentation

For a thorough explanation of all of RailClone's features, including procedures for many common operations, take a look at the Documentation.

11.4 Gallery

Get inspiration from some of the excellent work submitted to our gallery, and if you create an outstanding image or animation with RailClone (or Forest Pack) we'd love to see it. We're always on the look out for high quality material to feature on our gallery and social media , so if you

have work you think meets the brief, please send your renders to submission@itoosoft.com , or upload them to our forum . We'll share them on our networks with full credit and links to your website.

We hope you enjoy using RailClone and wish you the very best with your work. If you have any feedback, queries or suggestions please let us know.