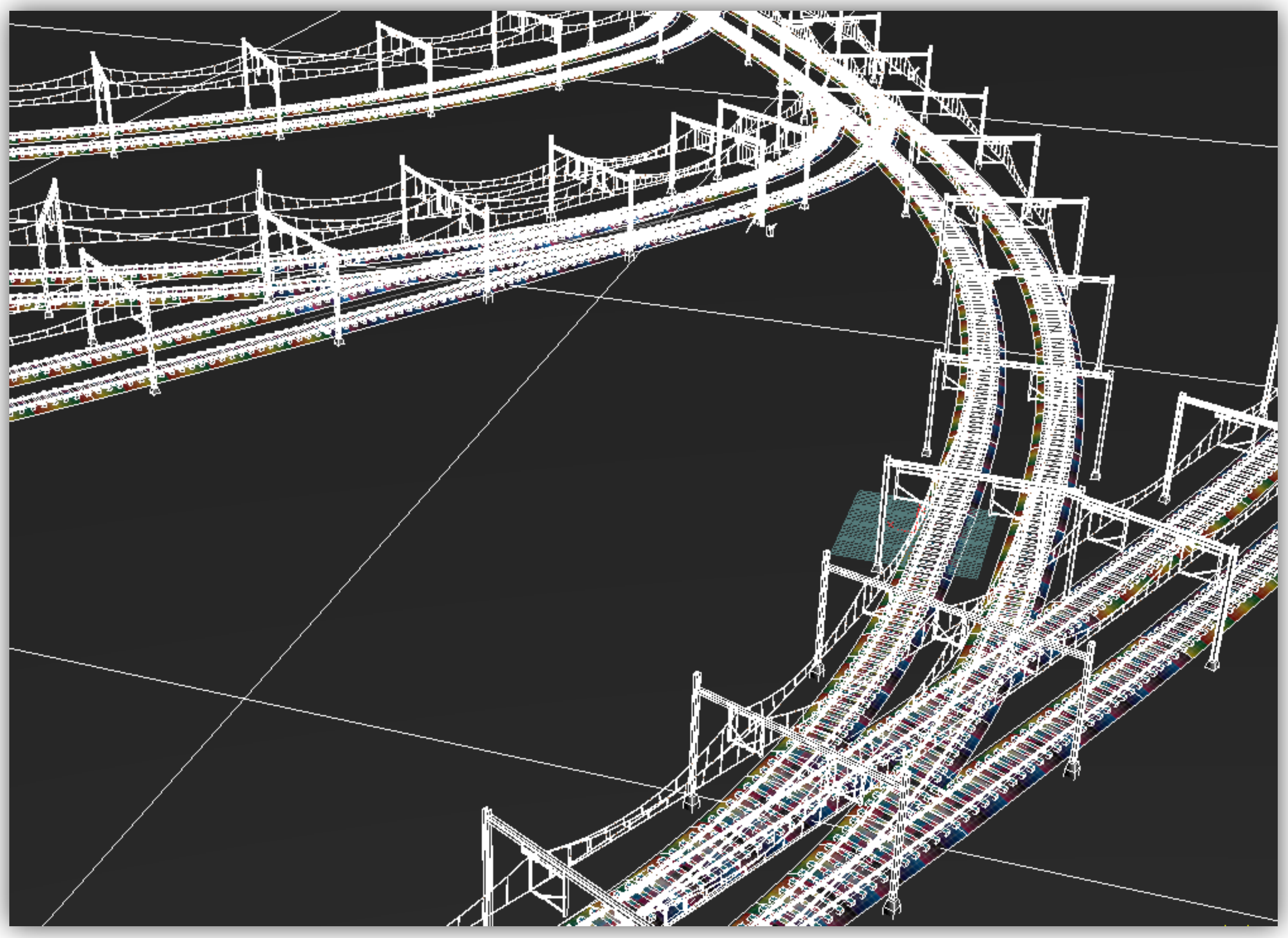
Specialization Dissertation: Procedural modeling

A Dissertation on the subject of procedural modeling of railroads and creating procedural railroad switches.



Name:	Freek Hoekstra
Supervisor:	Kim Goossens
Student ID:	072130
Track:	Visual Artist
Date:	20 June 2011
Period:	block C & D
Number of ECTS:	21

NHTV International Game Architecture and Design
The Academy of Digital Entertainment
Breda University of Applied sciences

Abstract:

In this paper a method is presented for creating curve-based procedural railway systems that automatically generate switches from any number of rough input curves, the goals being to minimize the amount of input required by the user.

To create the aforementioned switches, a spherical intersection is made at every curve intersection on the original input curves (which only need to intersect to work) a new set of curves are then created inside this gap, by using the intersection points between the sphere and the original curves together with a calculated intersection point.

Finally the procedure sweeps profile curves along the generated curves as well as copies objects along the curves in order to generate the final geometry, with no artists intervention required.

Table of Content

1. Introduction

As computer graphics keep increasing in visual quality the amount of artists needed to detail the world increases year after year.
Procedural modeling is a method of creating geometry that allows artists to generate high resolution geometry with minimal input, by making the computer generate the geometry according to rules, parts of the production process can be automated, enabling faster production times and greater flexibility.

Traditionally every change to the layout of the environment forces an artist to go back in and fix any problems that may have arisen. With the increasing level of detail (which can be observed, among others, in the Unreal series) this is becoming an increasing problem. While possible to manually update all models and environments constantly, there are numerous situations where this is extremely undesirable. For instance, manually exchanging the bolts of a bridge, or changing the length of its span, the latter likely requiring a complete remodeling of the asset.

As changes always occur in a development process, reducing the amount of work that needs to be redone is a constant priority.
By generating geometry by a set of “rules” instead of traditionally modeling and placing props by hand, the artists never have to throw away work. Modifying the input geometry will result in the computer updating the entire model/world automatically. For the bridge this means that with procedural modeling the artist only modifies the input bolt and changes the length of the input geometry to make the entire bridge update automatically in mere seconds.

Procedural technology is applicable to almost every process of game creation. For example, levels can be detailed procedurally allowing designers to make changes until the last minute without ever having to worry about artists having to redo or throw away work, allowing for more and longer iteration.

1.1 Previous work:

With the increasing demand of high detail environments, lots of research is being done on the field of procedurally generating terrain, procedural texturing and more recently, generating procedural layouts of cities.

On a smaller scale adaptive geometry between surfaces and along curves is being heavily researched. Also procedural buildings have become a common subject of research.

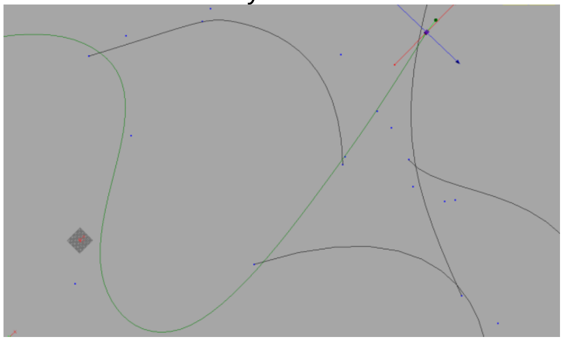
Fully procedural solutions for generating levels have been researched and used in commercial games like Diablo 1 and 2 and certain strategy games, up to entire galaxies as early as in 1984 by Elite.

1.2 Goals and challenges

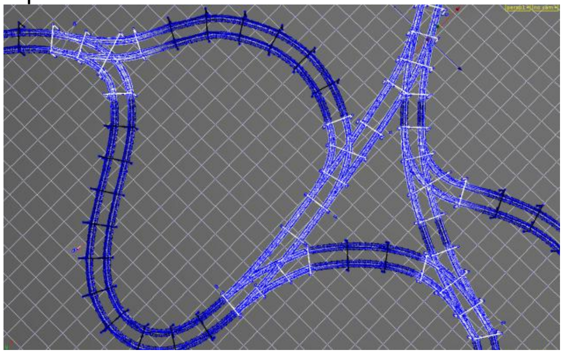
With the procedural railroad project the primary objective was to create a way to generate tangent intersections automatically, requiring no intervention from the artists. This paves the way for a very controllable environment creation tool by allowing the user to modify and control the shape and look of the procedure very precisely (something not always possible in a city generator). This enables a very fast workflow for creating procedural environments, but still allows for environments that very closely mimic the designers specifications without being too labour intensive or too random.

1.3 Input

For this procedural railroad system the artist has to deliver one or more curves that may or may not be intersecting. At every intersection a switch will be created and the entire railroad is detailed automatically.



Input:

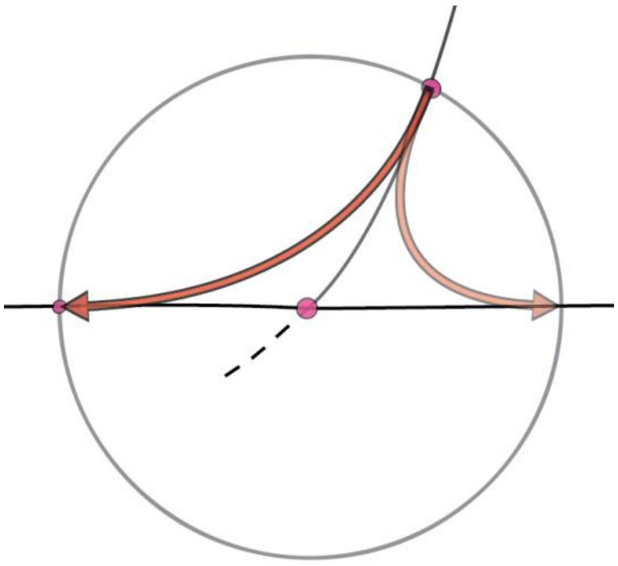


Output:

Additionally the artist is be able to control variables, altering the way the railroad looks

2. Curve generation.

The following image accurately describes the way the procedure creates the switch curves in a simplified manner. The following section will describe roughly how I came to this solution and how it works:
To create the switch curves are required for the railroad. First the two original input curves are intersected, creating an intersection point



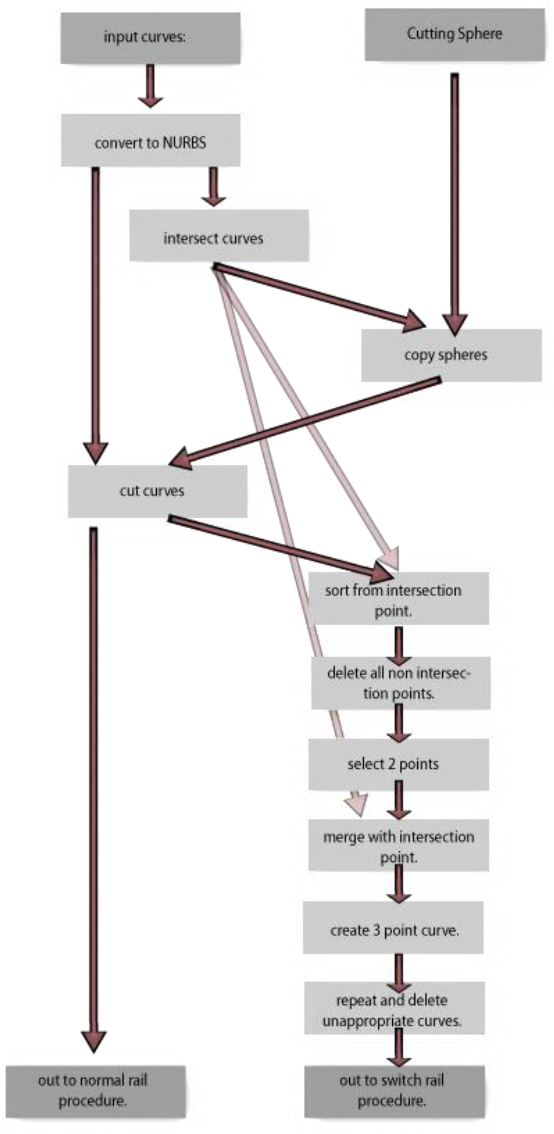
(purple dot in the centre).

Onto this point a sphere is copied, which is then used to cut the original curves. The points at which the sphere intersects the curves are selected and used as a start/end point for the newly generated curves (the red arrows).

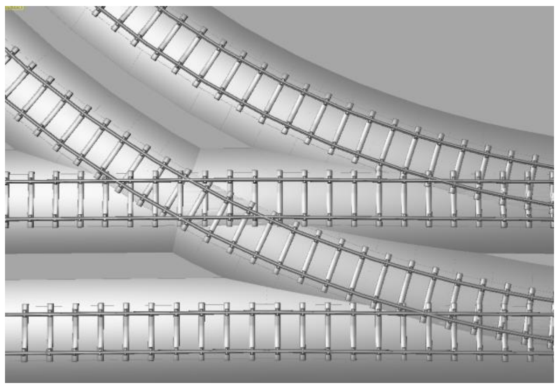
Two of the intersection points are selected by deleting one of the three. These will form the start and the end point of the curve. Merging the two points with the original intersection point (purple) provides 3 points on a row. A new Nurbs curve is created. That results in an (almost) tangent curve. The curve is checked for acuteness and if this is the case removed. The process is repeated until all curves are processed and the two correct curves remain.

2.1 The graph:

Another way to visualize the procedure is by looking at the general tree setup. This is a simplified version of the actual graph used to create the curves but provides a clearer image of what every portion does:


input curves:
Cutting Sphere
convert to NURBS
intersect curves
copy spheres
cut curves
sort from intersection point.
delete all non intersec-tion points.
select 2 points
merge with intersection point.
create 3 point curve.
repeat and delete unappropriate curves.
out to normal rail procedure.
out to switch rail procedure.

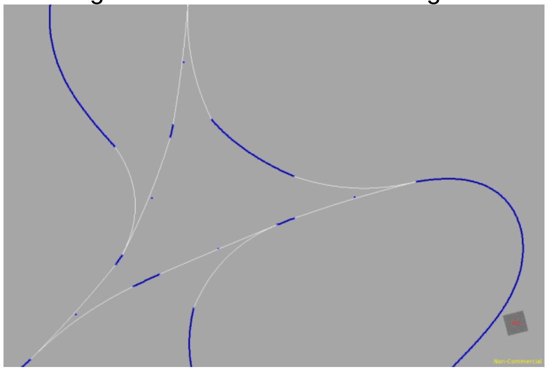
The graph has two outputs. One part will feed the normal “standard” rail procedure, while the other will feed into the more complex switch detailing procedure. The two procedures are separated because the switch procedure is far more complex as it must among other things prevent intersection from occurring between Ties as demonstrated below:



Note the intersection between the beams.

2.2 Resulting curves:

After the curves have been processed the resulting curves look similar the image below:



The curves depicted above form the basis for the rest of the procedure. They are utilized to generate and or place almost all parts of the railroad. The blue curves are the original curves as drawn by the artist minus the intersections. These will receive the normal detailing procedure.

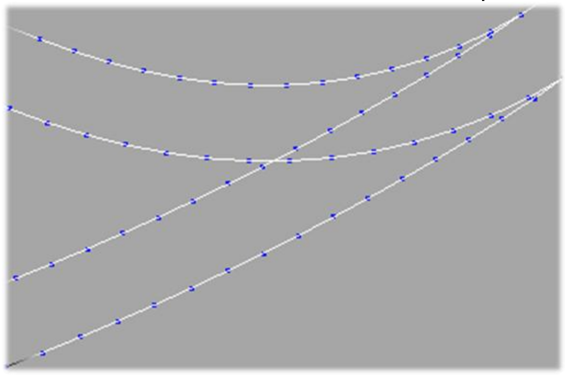
The white curves are switch curves as generated by the procedure within the “gap” inside each spherical intersection. It will receive more complex treatment as detailed in chapter 3.

2.3 Detailing the track

By continually using lines to produce and place the assets the procedure is kept controllable at all times even though the result is complex. Changing distance or size of an element is always easy as it just means changing one variable: the length of the corresponding line.

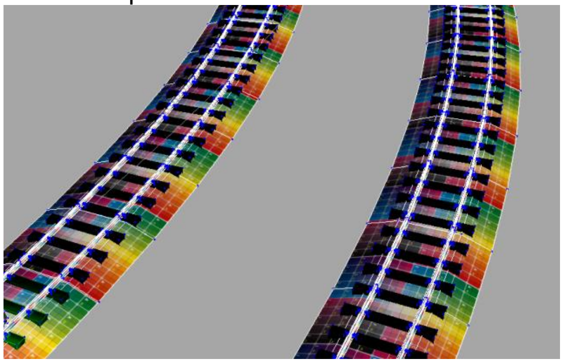
2.3.1 Creating the number of tracks

After using a refine node so the curves are optimized unnecessary points are removed based on the curvature, the curve is swept.



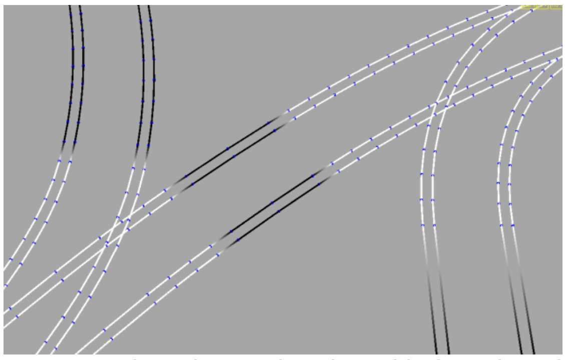
The distance between the lines is controlled by the width of the line that is being swept. After sweeping the lines a skin node is used to connect the points and create continuous lines.

Along these curves a number of profiles will be swept to create the ballast, and after resampling to an even distance ties are copied onto each point.



Swept ballast and copied ties:

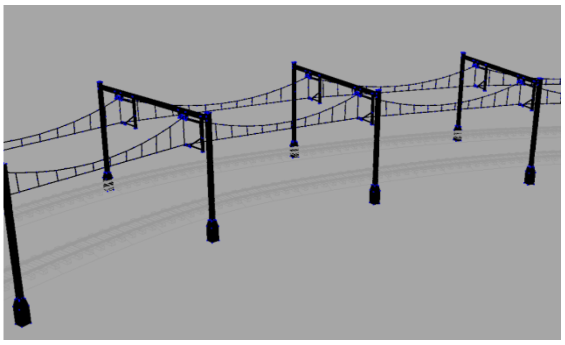
2.3.3. Creating the rails



To create the rails another line with the railroad width is swept along the curve. it is finished by sweeping a profile curve (the cross-section of a rail) along the length of the curves.

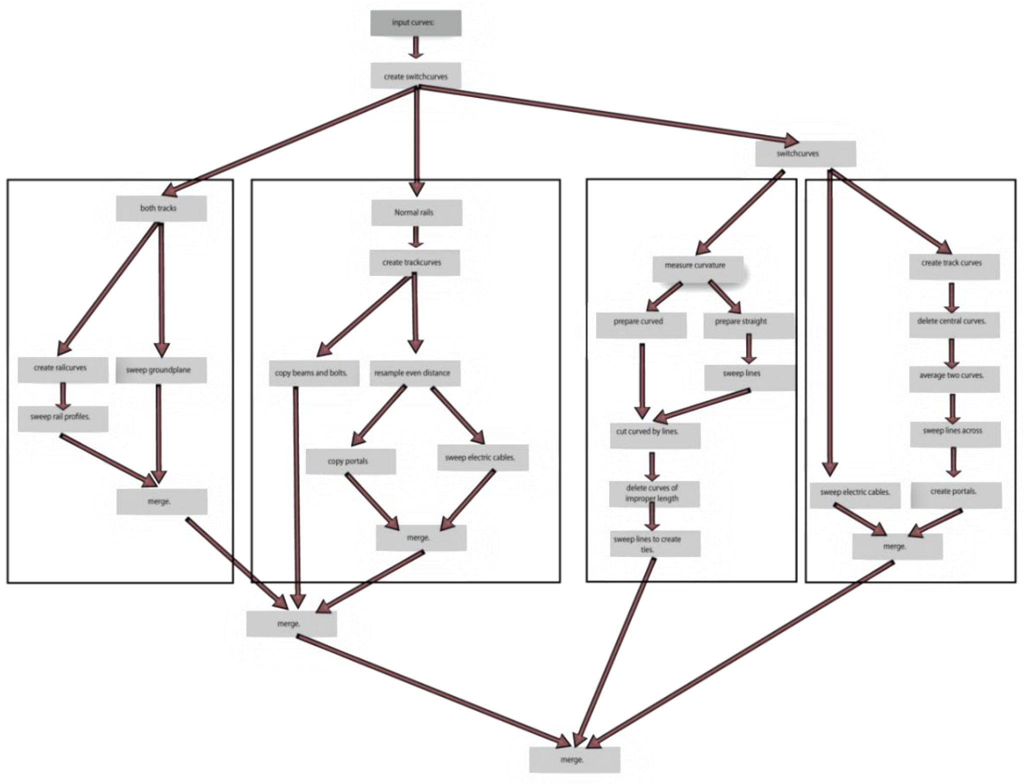
2.3.2 Creating the electrics

To create the portals the curve from chapter (2.2) is resampled to get an even distribution of points along each curve. At every point a portal gets created, and across every span an electric cables are created



Example of portals and electrics. This process will be elaborated on in chapter 3.

3. Breaking down the network



After explaining briefly how the objects are created on the normal rails the more complex switches will be treated. Above is a simplified overview of the entire procedure, starting from the input curves, all the way down to the final merge. The procedure can be split up into 4 modules from left to right**:**

1- creating the railroad curves (top nodes).
2- creating the rails and ballast.
3- creating the ties, portals and electrics for the normal curves.
4- creating switch ties.
5- creating the electrics and portals for the switch curves.

As mentioned the curves are treated separately, and at the final node the results merged. The top node (create switch curves) is the one covered in chapter 2.1. Not all nodes will be covered due to the amount of information processed. However, every node represented above is named by the action that in reality is being processed by a collection of nodes to give a better understanding of the procedure.

3.1 Creating the portals and electrics

The following part will discuss in detail how the portals were created starting from the input curve. The portal creation is different from the most other parts of the procedure as it is a different type of structure and therefore cannot be swept.

3.1.1 Input curves:

To create the portals the curve generated at chapter 2.2 (create switch curves node) is utilized. There is a single portal for any number of tracks thus 2.3 is only used for the electrics.

The curve is resampled to an even distance using a for loop and a resample node. It determines the length of each curve and divides it by the desired length variable.

arclen("../inputcurve",0,0,1)/ch("../resample_distance")

This generates a curve with equal size segments along each individual curve, preventing “double placement” near the ends of curves.

3.2 Portal Creation

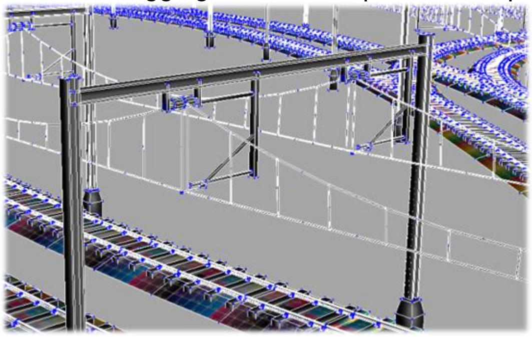
On each point of the curve a copy of the portal is placed. The height and width of the portal are determined by the variables used to create the rest of the railroad, freeing the artists from having to manually synchronize the size of the portal with the rest of the procedure.

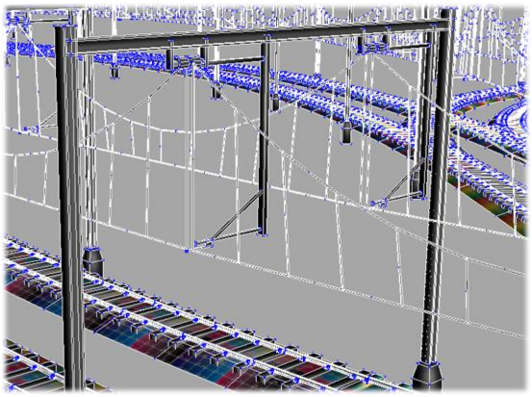
For instance the width is determined by:
*track_width + ((rail_width*0.5)*number_of_tracks.)+offset*

This creates the railroad while allowing the artist to control the offset, providing full control with just one variable.

3.2.1 Reversed vertical setup

For the height a similar system is utilized.
The procedure is built from the ground up.
Not the other way around, as increasing the height of the beam would result in the train not getting power any more. To determine the height of the electric cables the height of the train is added to height of the rail, and to that the cable sagging the beam is placed on top.



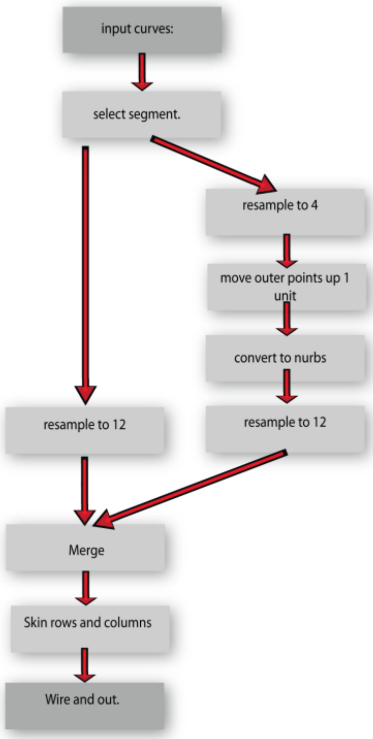


More cable sagging causes the rest of the system to be raised making sure the system always remains intact. The system always adjusts automatically to any changing variable without compromising another. For instance, if the sagging of the cables increases the support beam moves up and the suspension structure adjusts to compensate, making sure the train still fits below.

To finally finish the portal a ray SOP is used from the two outer points and projected down towards the ground. Lines are drawn to the ground (or 0 height if none is present) and a profile swept along to draw the pillars.

3.3. Creating the electric cables

Curve (2.2) is taken and segmented with an even distance resample locked at the same settings as the portals. This way the cables always match up from portal to portal.



To create the electric cables a single section of the curve is selected to be labelled wire. The Wire curve is resampled to have 3 sections (or 4 points) and the middle two are dragged down. Using a convert the polygonal curve is converted to a NURBS curve, making it smooth. The curve is resampled again to have more points and a transform is added with controls for scaling (more or less sagging).

Furthermore another procedure is also run on the wire. It is resampled to have the same number of points as the one treated before.

The two are merged together, and using a skin node the lines are connected to form a mesh. Finally a wire SOP is used to give thickness to the lines and the electric cable is complete.

3.4 Creating the switch portals

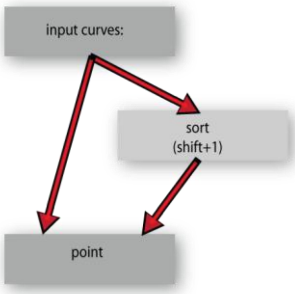
The switch portals are setup in a similar way to the normal portals. The vertical setup is identical, but creating spans that automatically stretch across spreading tracks required some different solutions. To create the spans first the switch curves need to be sorted and the middle two need to be removed. After this is complete an average curve can be created. This is done by using a Point SOP with the following expression:

$TX2*0.5+$TX*0.5
$TY2*0.5+$TY*0.5
$TZ2*0.5+$TZ*0.5

This averages the points and creates a curve which can be used to copy a line onto. These copies are intersected with the outer switch curves (generated above) to generate the locations where the pillars will stand.

3.4.1 Extending the curves

This is the basis of the automatically increasing width. However to prevent the pillars from intersecting with the railroad itself the curve needs to be extended. Therefore a normal is created on each point which is later used to extend the curve.



To generate the new normal pointing along the direction of the curve Inside the point node the following expression is used in the normal generation tab

$TX2-$TX
$TY2-$TY
$TZ2-$TZ

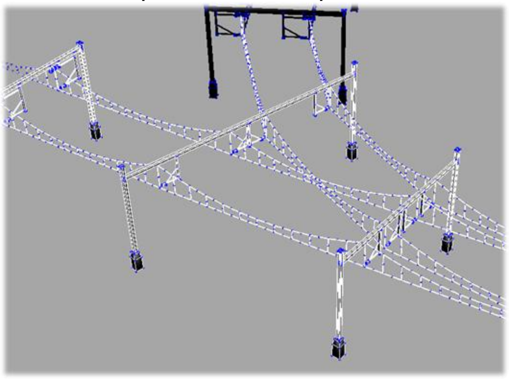
As the point numbers have been switched (point 0 has become point 1 and vice versa) subtracting the position of point 0 of input 1 with the position of point 0 of input 2 results in a vector (stored as a normal) between the two points. The normals are normalized (they are presently not unit length) and the points are displaced outwards along the negative direction (the normals are pointing inwards not outwards) by another point node as per the following expression:

$TX-$NX((ch("../../track_dist")/2)-ch(“../portal_offset”))*
$TZ-$NZ((ch("../../track_dist")/2)-ch(“../portal_offset”))*

The track distance is halved as the displacement happens on both points. Therefore half a displacement on both sides suffices. The portal offset is driven by the same offset and allows the artist to increase or decrease width of the portals with the same control.

3.4.2 Placing the cable suspenders

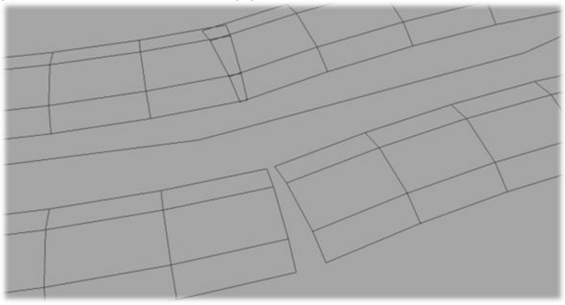
Not only must the width of the portal now be calculated, the locations where the suspension structures must be placed to properly suspend the electric cables must also be computed.
Using the original input curves of 2.2 and intersecting them with the curves generated for the portal beams, intersection points are generated. Normals are created and on the points the suspenders are copied.



Note: The second portal is wider than the first.

4. Curve tangency

Although the basis of the procedure is now functional and generates a curve interpolating between the 3 intersection points there is still a tangency issue. It is normally not very visible when just displaying the curves themselves, but when multiple tracks are swept along it the problem becomes apparent:

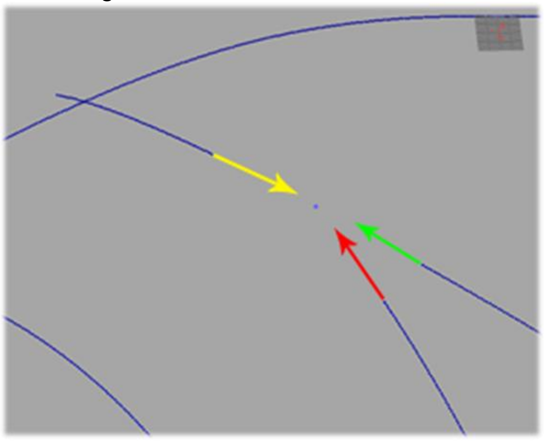


A fairly extreme example of a tangency issue.

The tangency issue occurs because the 3 point curve generated in chapter 2 does not take into account the direction (last normal) of the original curve. Therefore the curves no not have the same normal. The curve processed is only tangent if both curves enter in a straight line towards the intersection point. This is a rare case as they in reality are often curved, resulting in an inconsistency in the normals, which finally results in the issue displaced on the previous page.

4.1 Generating tangent curves

To ensure tangency, a new intersection point is calculated based on the direction of the last normal of each point on each curve. To calculate the intersection two straight lines that extend along the negative direction of the last normal of each curve are created (as per the method discusses in 3.4.1). Being straight they are perfectly tangent with the last point of the input curve. The lines are intersected providing the point in space where the curves would have intersected would they have continued on in a straight line.



This matches our constraint of the 3 point curve being tangent only with straight lines. As the intersection point through which it interpolates is now based on straight lines the resulting NURBS curve is also tangent at both ends.

4.1.1 Creating the curve normals

To determine the normal of the last point of every curve we splice in from the original spherical intersection volume break and the point sort (as discussed in chapter 2.2), which is still sorting from the original intersection point.

Deleteing every point except for the first 6 points: 3 primitives with 6 points remain.

A for loop is created at which point we delete the two other primitives and create the normals as described in 3.4.1.

4.1.2 The lines

In order to create the tangent lines the secondary point is deleted and the first point is displaced along the normal:
*$TX-$NX*ch("../switch_size")*2*
*$TY-$NY*ch("../switch_size")*2*
*$TZ-$NZ*ch("../switch_size")*2*

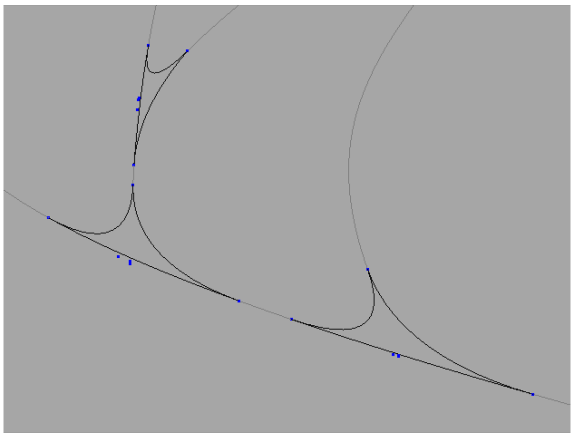
This moves the point exactly to the opposite end of the spherical intersection, and by using the switch_size attribute the line scales with the size of the switch selected by the user.
The coordinates are transferred to a line and the loop ends to reveal 3 intersecting lines.

Deleting one line based on the stamp number the intersection is calculated between two of them and the intersection point is stored. This point is the point where the two original curves would have intersected were they to continue straight from the last normal.

4.2 Creating the final curves

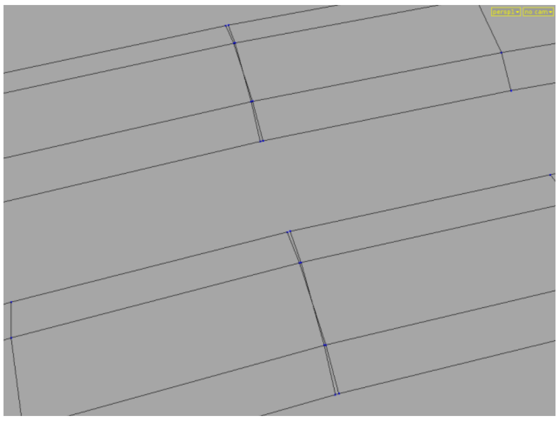
A new curved line can now be generated. 3 points are taken from the volumebreak / sort (spherical intersection) and two points selected. These points are merged with the new calculated intersection points and sorted (shift +1) resulting in a usable 0-1-2 point order.

Finally a point SOP is used to transfer the coordinate to a 3 point curve. Because the new intersection point is now placed exactly along both normals the tangency problem has been resolved.



Although the previous method yields similar results the generated curves are now much closer to tangent. After using the dot product is used to delete the acute curve and the results are swept. It is visible that the result is much more accurate.

The result is as before a system that can generate switches at any given angle, but with much better matching tangency, as is clearly visible in the image below:



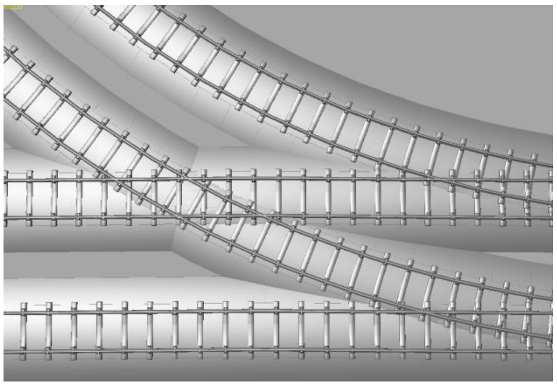
To fix the last bit of overlapping a Fuse node is used, (not used on the image above for demonstration purposes) fusing the points together and creates a seamless mesh.

5. Placing the ties

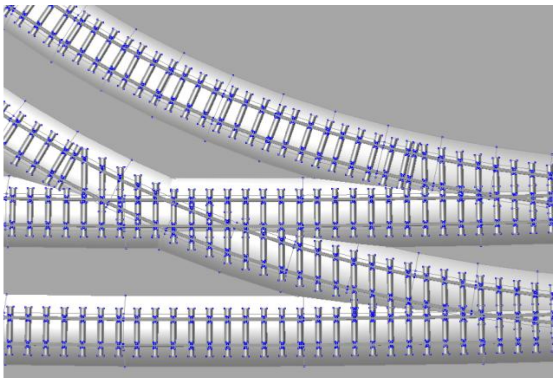
As before there is a great difference in object creation and placement for the "normal" rails and switches. For the normal rails placing the ties is a simple matter of copying the geometry onto the points.

To replicate real-world construction of ties on switches however, the ties that would normally intersect (if just copied on every point) need to be replaced. A new procedure for placing must be designed taking into account intersections and prevent any intersecting geometry.

The following image shows the situation as is: intersecting geometry, as the geometry is placed by resampling the curve and copying a tie on every point.



Intersection needs to be prevented and replaced by beams crossing the entire span, resulting in the image below.



Ties stretching across to provide support without intersecting, also stretching when required (the splitting off rails).

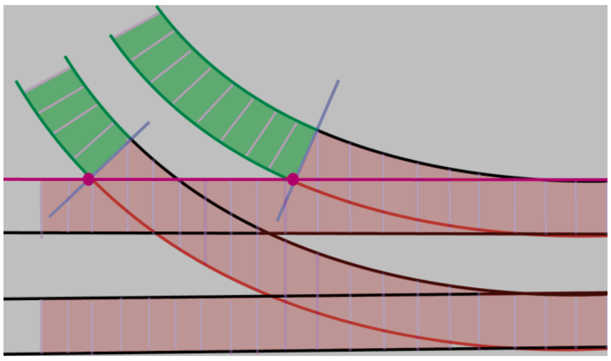
5.1 Preparing the curves

After the selection is made on what curves to work on (by for loop) the curvature of both curves need to be calculated. The straighter curve will be used to project points from onto the curved track.

The curvature is calculated by sweeping a line across the curve and comparing the length of each section in a VOPSOP. The values are summed up and stored as a colour. These are averaged, and the curve with the higher value, therefore the most curved, kept separate.

5.1.1 Isolating the proper curves

The ties in the switch are treated in two ways, the parts that have yet to intersect, and thus are aligned with the straight curve (red), and the parts that have intersected, and will not intersect again bending along the track (green).

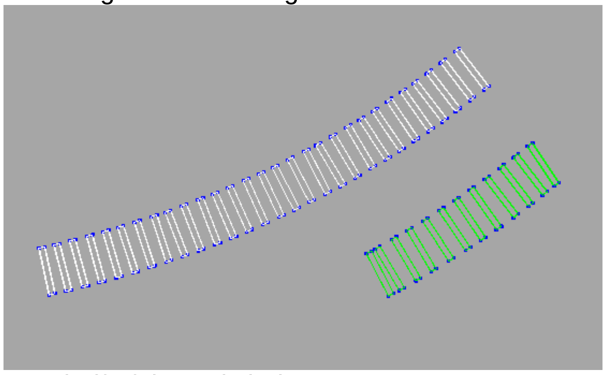


To prevent any points to be projected on the rail that will not be intersecting any more

every second curve from the curved track gets selected (red curves) at the intersection with the inner most straight curve (violet).
The blue line is created (at 90 degrees to the normal of the curved curve).

The other (curved) curves are cut by this straight line to create a straight cut. After this cut no beams will stretch towards it preventing slanted ties. This means the green side will not be using the complex positioning, as it will get ties that follow the direction of the track. The curves on the green side are resampled to get a same number of points on both sides and thus straight connections. An add node used to create the lines between them.

Resulting in the following ties:

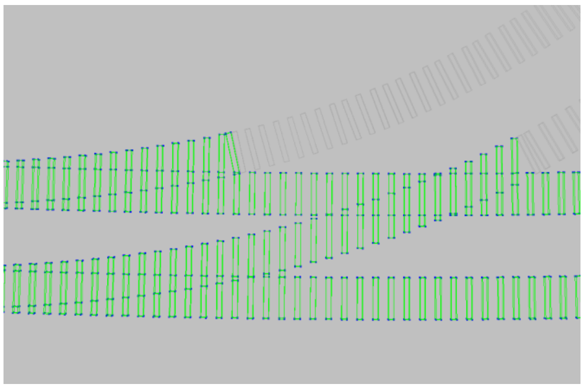


One half of the switch ties.

5.2 Creating the Ties

To finish the other half of the switch ties, the straight curves are selected plus the other side of the curved curves (the red side). The straight curve is resampled and lines swept along the curve, and all curves are curvesected with the lines.

An add node is used to connect all the points, and any curves with too long a length are used to remove any unwanted geometry. Finally a grid is swept along all lines to create the ties, resulting in the following image:

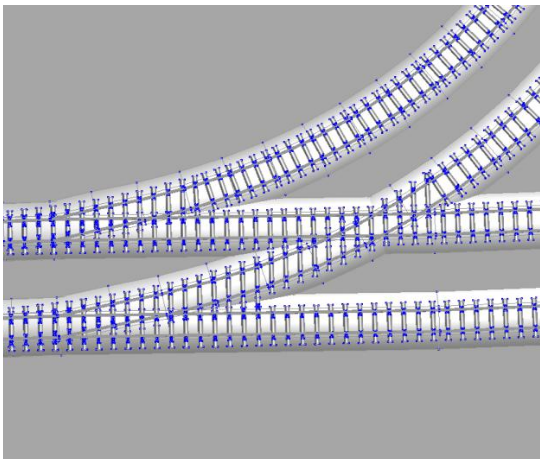


The other half of the tie procedure

The bolts and details are added, and both sections of ties are combined. The result is a system that generates proper supports under any angle.

5.3 Combining the results:

The two sections of the switcht ties are combined and the solution below is created.



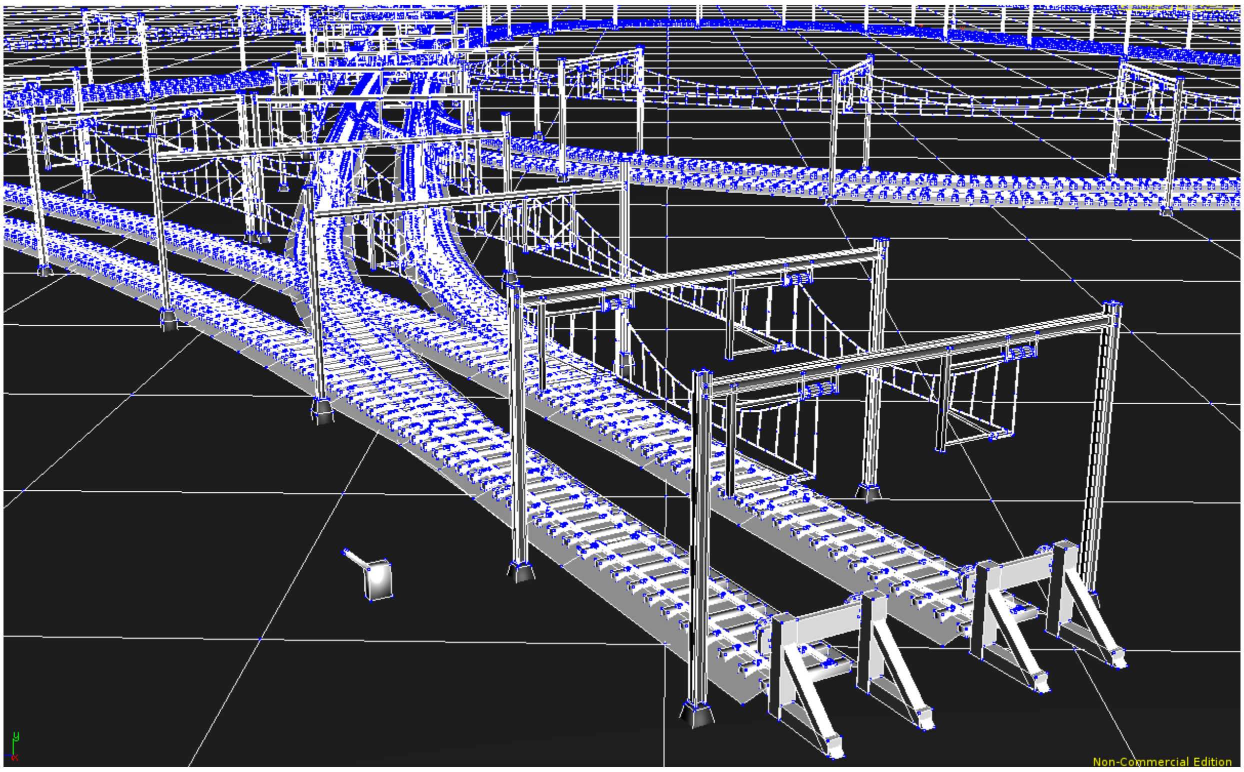
The final result of the switch tie procedure

The switchties are merged with each other and with the rest of the procedure .

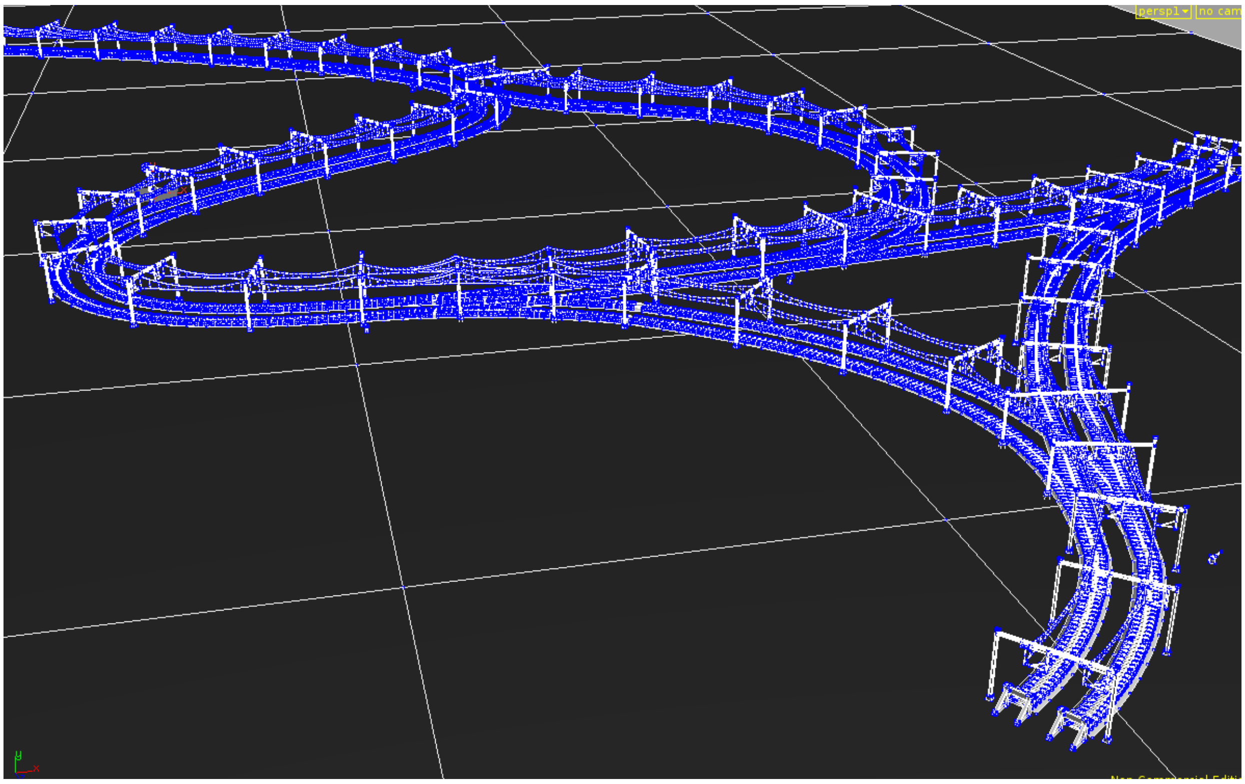
The results will be displayed in the following chapter, displaying the procedure processing various settings and utilising various input curves , enabling an artist to set up an entire railroad network in minutes. A task that would be very labour intensive to do by hand and would take several days depending on the scale of the railroad.

6. Combining the Results

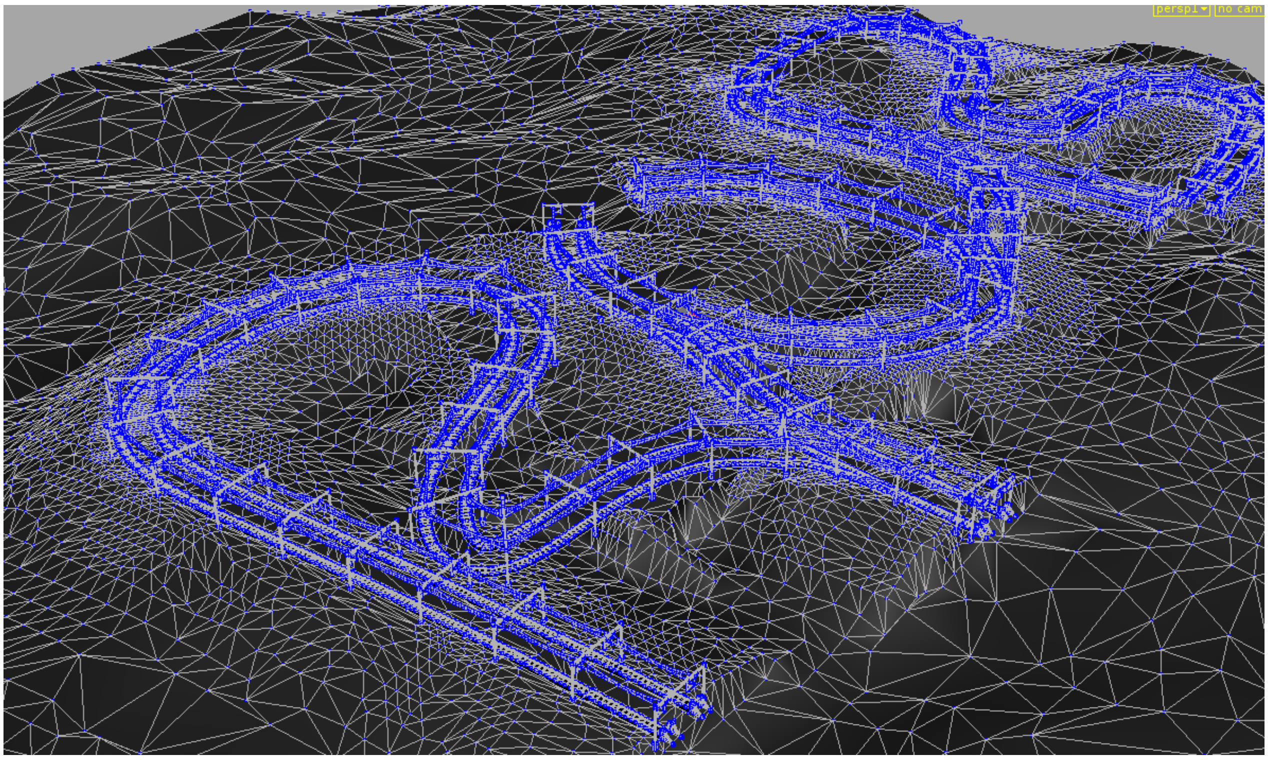
A few screenshots of the complete procedure in action:



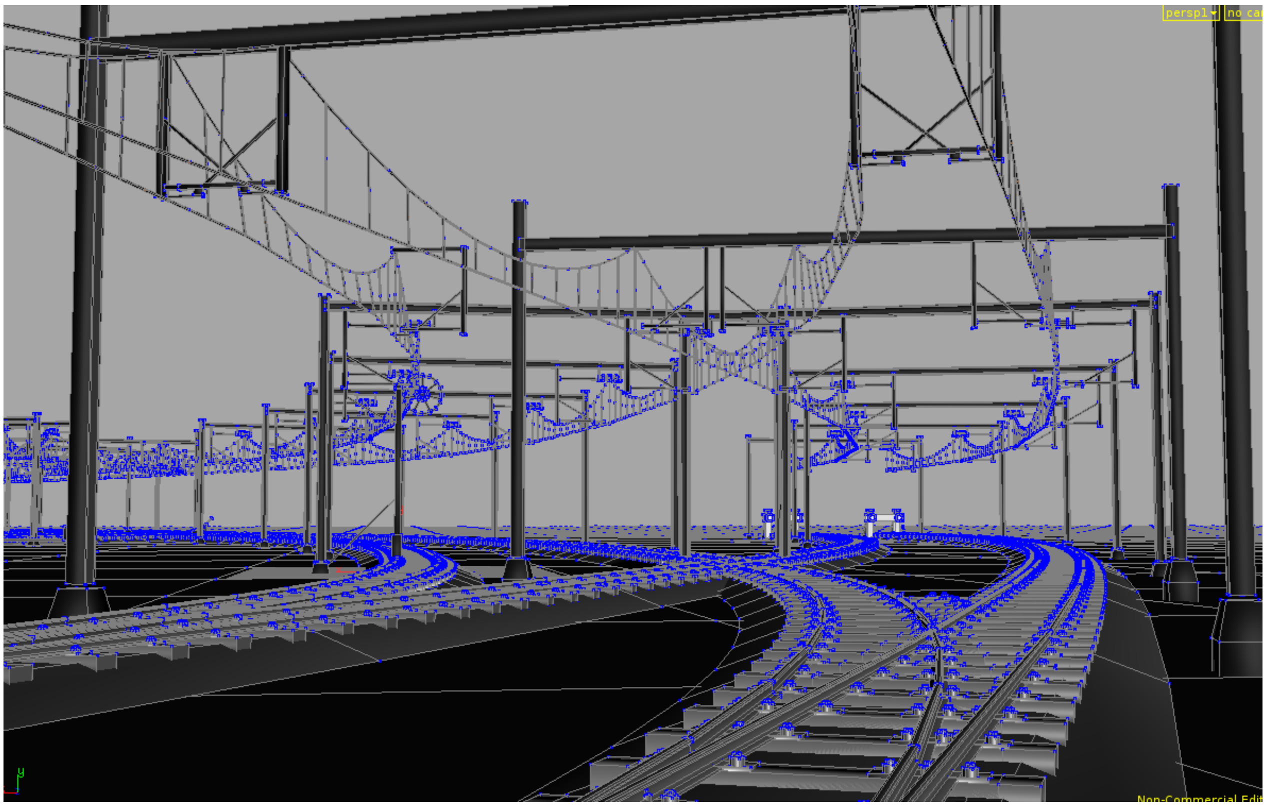
At the endings automatic train stops are created.



A birds eye view.



A test with terrain implementation. The tracks curves created by my roommate to test for stability.



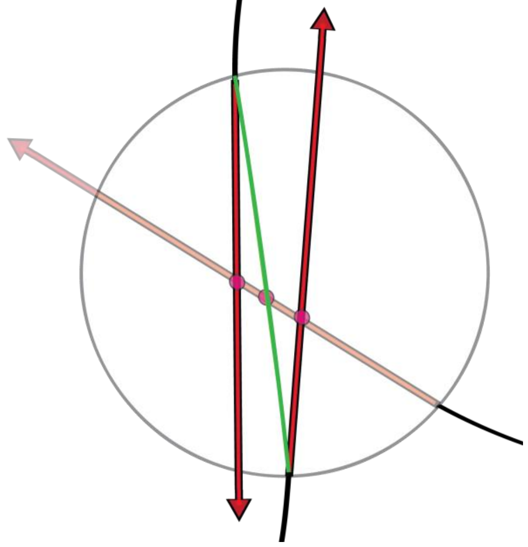
A close up of the rails revealing the bolts.

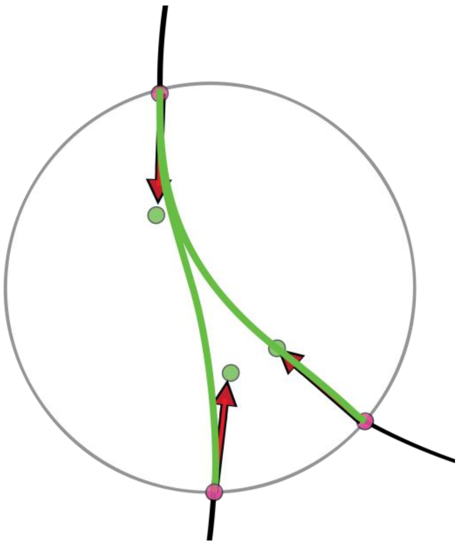
7. Limitations and constraints

Below are listed a number of exceptions in which the procedure does not respond appropriately, followed by a short explanation why.

***Continuing curves**:*
As of yet the procedure does not support intersecting curves that both continue onwards. Practically this means that at an intersection one curve must continue and the other must end. Support has been removed for continuing curves, as the procedure proved unable to accurately predict what curves the user wanted to keep and which ones to remove. Further research might improve upon this area as comparing angles could potentially provide proper results. Self-intersection is also not supported.

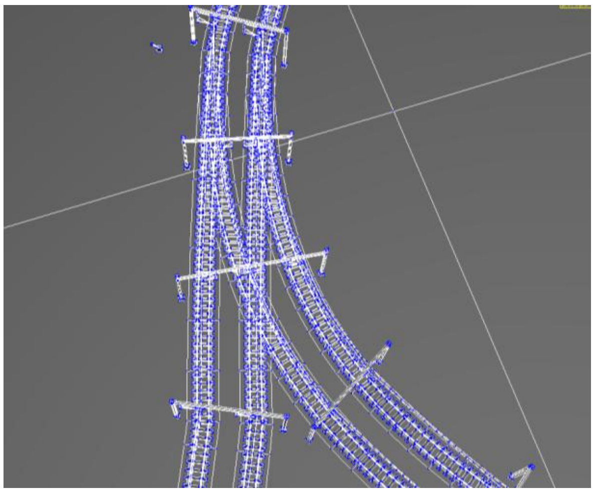
Tangency:
In the tangency operation there is a small chance of failure. With a certain setup of curves it is possible for the projected tangents to never cross (see image below). To solve this issue currently an average of all intersection points is used as a backup. Also the user can choose to use the simple intersection method which sometimes yields better results (chapter 2).

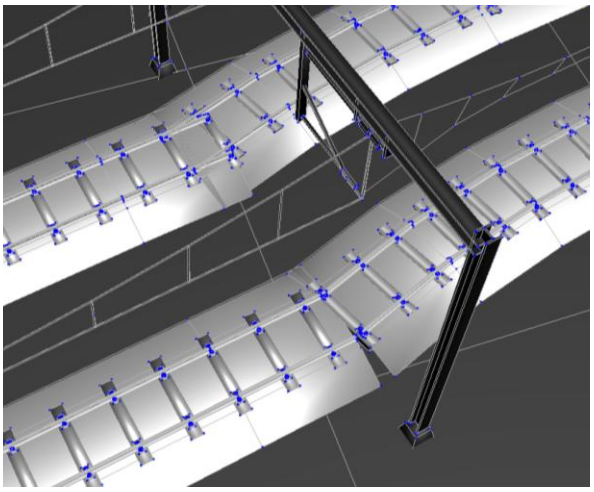

No intersection, therefore an average is used.


The optimal curves, resulting in an S-shape.

A true proper solution should be researched, possibly into the lines of a 4 point curve as per the image to the right. Using the vectors instead of an intersection point, another solution is to make the curves coming into the circle always align to the normal of the circle.

Below is an example of the current solution and the error, Note the “kink” in the rails..





7.1 guidelines

To utilize the Railroad procedure some things to be taken into account as a number of conditions do need to be met to successfully generate a proper railroad. These conditions, although for the bigger part also applying in real life, are the subject of further research in order to further improve the usability of the asset.

A few basic guidelines while using the procedure are:

- at every intersection one railroad must continue another must stop. crossing intersections are not currently supported. also ending both curves is not supported, nor is self-intersection

- two spheres for switch creation may not intersect where they cut a track, as two switches cannot be generated in the same spot.

- the tracks must be fully split before they exit the spherical intersection to ensure proper Tie placement. (outside the sphere the procedure falls back to normal tie placement)

if a problem with tangency occurs switching the curve generation method can help. However making more space for the switch and making sure the normals intersect solves the problem.

8 Literature

8.1 Tutorials

Procedural road creation ~ by Kim Goossens
Website; Video Tutorial: 3 parts.
Publisher: CMIVFX (03-03-2010)
Language: English
Source: www.CMIVFX.com

SIDEFX.com ~ by Various artists
Website; video tutorials
Publisher: SideFX Software
Language: English
Source: www.sidefx.com/index.php?option=com_content&task=blogsection&id=14&Itemid=132

Houdini Tutorials ~ by Peter Quint
Website, Vimeo Channel
Publisher: -
Language: English
Source: http://vimeo.com/channels/54102

8.2 Technical

Houdini Help File ~ by SideFX
Houdini source.
Publisher: SideFX
Language: English

8.3 Reference

Railroad construction theory and practice ~ by Walter Loring Webb
Hardcover: 312 pages ~ excluding tables. (196 pages)
Publisher: BiblioBazaar
Language: English
ISBN: 055996353X
ISBN-13: 9780559963537

Video game costs ~ VGcharts
Website; Internet article
Publisher; VGcharts
Language; English
http://vgsales.wikia.com/wiki/Video_game_costs

Special thanks go to:

Kim goossens:	*for introducing me to Houdini.*
Andrew Paquette:	*for proofreading this dissertation*
Tessa el Miligi:	*for proofreading this dissertation.*