Automating the Transportation Design to Simulator Model Process

SAFETY RESEARCH USING SIMULATION
UNIVERSITY TRANSPORTATION CENTER

Shawn Allen, BFA, AA
Engineering Coordinator
NADS
The University of Iowa

Vincent Horosewski, BS
NADS
The University of Iowa

Automating the Transportation Design to Simulator Model Process

Shawn Allen, BFA, AA
Project Mgr., Transportation Visualization
NADS
University of Iowa

Zuoyuan Zhao
Undergraduate
Mathematics, Computer Science
University of Iowa

Vincent Horosewski
Application Developer
NADS
University of Iowa

Adam Kueny
Post-graduate
Engineering
University of Iowa

A Report on Research Sponsored by SaferSim

July 2017

Table of Contents

List of Figures

List of Tables

Abstract

This development project was proposed to the USDOT in the SAFER-SIM proposal under *Theme Areas: 1. transportation safety affecting vehicular roadway users, and 2. enhancing roadway design processes*.

The purpose of this project was to refine software tools and algorithms that automate the conversion of transportation design models into NADS miniSim^{TM} driving simulation environments.

Outreach to miniSim users was made in hopes of developing a consortium of simulation laboratories interested in working with Departments of Transportation research or design activities, including demonstrations of proposed designs. Several sites have in the past expressed interest in building or converting tile model environments from transportation design models. A small number of models were submitted by one site, but the lack of necessary design model attributes and inability to provide them required the creation of smaller design models than would have been available through the consortium.

Converter improvements were made to the existing tool by integrating XML and spline libraries and algorithm improvements. Several conversions were made from transportation design models into driving simulator models. For purposes of this development, driving simulator models are defined as the visual environment model and associated logical meta-data necessary for driving the model using the NADS miniSim driving simulator.

As part of the project, different simulator formats were investigated to identify 3rd party simulator formats that might be suitable for the converter to support in addition to NADS miniSim formats. Such an approach could benefit the broader driving simulator and transportation research communities by expanding their capabilities to use design models.

Developing a robust converter requires that different types of roadway design models be used to exercise converter algorithms and work flow. The test cases vary in complexity and scale and are detailed in section 3 and include simple single roadways, hills, curved roads and intersections.

1 Project Overview

This section provides an overview of the project and subtasks undertaken during the performance period.

In addition to software development, an effort was made to create a virtual consortium of laboratories that we believed to have a relationship with local departments of transportation or who had expressed interest in the capability of simulating roadway design models.

A number of enhancements were implemented on the converter tool to improve performance. Standard libraries were integrated to provide robust solutions to smoothing algorithms and parsing/access to LandXML data.

The converter was exercised on a number of test cases with varying complexity, including straight, curved, hill, intersections and a small city-like network of roadways.

NADS test files were created using AutoDesk Civil3D^{TM} version 2018 and stored as DWG and also exported to LandXML. The file format LandXML retains document design elements that permit older versions of Civil3D to open later version design documents while retaining the ability to interact with the model. This is an important consideration since design models are tightly coupled to file formats, and the native DWG format is not backward compatible. Industry standard design software (Bentley Microstation and AutoDesk Civil3D) are able to import and export LandXML files.

Test cases submitted to NADS were reviewed using Civil3D. Design files missing external references and invalid or missing references are problematic for processing.

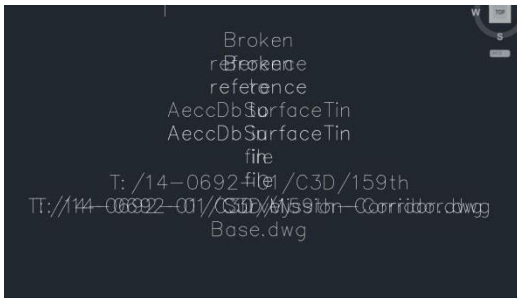

Figure 1 Test file missing associated TIN file

1.1 Converter support for 3rd party simulator formats

As part of the project, an investigation was conducted to determine which simulator formats might be appropriate to support by the converter tool. The two most widely used research driving simulators were determined to be OpenDS and STI^{TM}.

OpenDS (Open Driving Simulator) is based on the JMonkey^{TM} game engine which uses a proprietary, binary file format to store database information. Although converter could support this format since the project is open source (and therefore access is possible to source code), it was deemed to a poor choice due to the necessity of decoding this file format causing delays to other more critical project efforts.

STI encodes both scenario and database information in a single text file. Furthermore, the software imposes severe restrictions on the sort of changes that can be made to the visual environment in contradiction to the way the NADS simulation architecture permits run-time environment changes. This format was therefore also deemed not a good candidate to include in the NADS converter.

1.2 Collaboration Outreach

Twenty organizations were contacted to participate in a virtual consortium by lending their expertise and DOT design models as test cases for the converter tool. The objective of the consortium was to provide a variety of test cases for processing, and to create a pool of simulator models from these test cases that would be redistributed to each site that submitted design models. Additional participation was possible for laboratories that had no design model to submit, but wished to participate by nominating design model candidates.

A website was created with information about goals and participation guidelines at: http://safersim.nads-sc.uiowa.edu/converter/

Each model submitted for inclusion in the project was to be considered proprietary to the submitting lab or agency with the understanding the resulting simulator models would be shared among the consortium members.

The converter tools generate simulator-specific resources as well as visual model files and textures which are broadly supported by most 3D applications. These model files and textures can therefore be used on simulator architectures other than miniSim, and would theoretically appeal to a broader research and design community. Therefore the participation of laboratories that use simulators other than the NADS miniSim was possible.

1.2.1 Collaboration Result

Only one site expressed interest in the project and submitted 2 model candidates. Both models were from 3rd parties, not originating with the miniSim laboratories. Both models did not contain required design elements necessary for the converter, and despite several iterations we were ultimately unable to use these models for test cases.

A follow-up survey was sent to each candidate lab to identify reasons for non-participation. Lacking more detailed information it seems reasonable to believe the participation and re-distribution guidelines may have contributed to a decision to not participate. However, feedback received to date indicates that the goals of this project did not align with laboratory research, and one response indicates a perceived incompatibility (the lab uses a different simulator than miniSim) – despite the consortium website information specifically indicating the converter produces a generic visual model.

In retrospect each site should have been pursued individually and the points expressed clearly how participation could benefit all contributors.

1.2.2 Collaboration Test Models

Two test models were submitted for processing. However, these test models did not contain attributes necessary for conversion and updated design models could not be made available for processing.

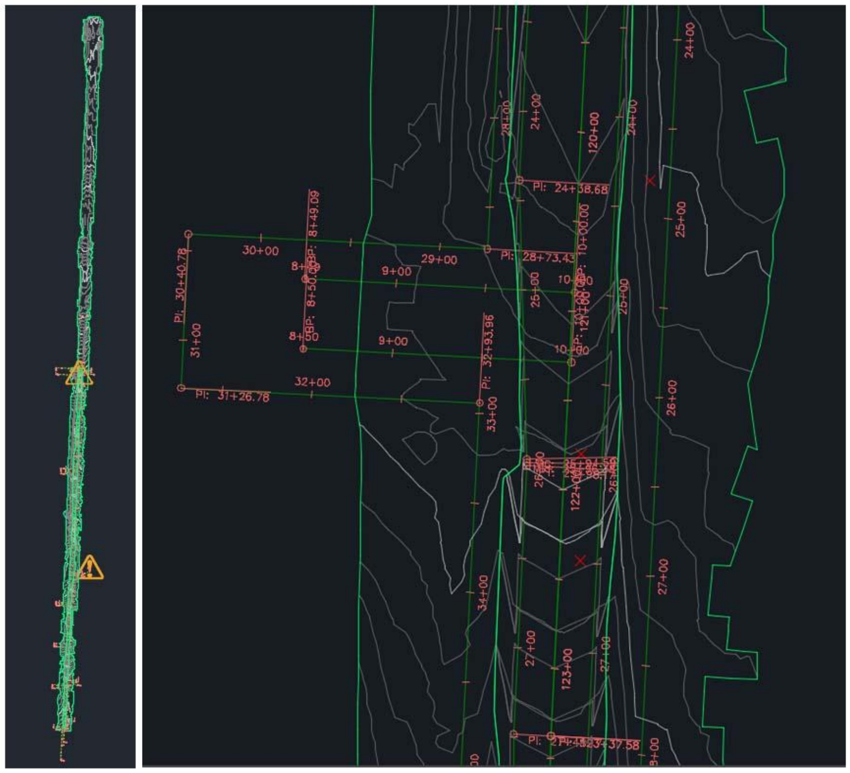

Figure 2 Test case 1 (left: site, right: zoom detail)

The test models were missing key attributes (i.e., breaklines) that had been used as hints to extract proper road geometry from the data included in the design file.

1.3 Converter software enhancement

The converter tool has been enhanced from previous versions with the addition of curve smoothing algorithms to generate smooth curve corridors within intersections. Performance has been improved by integrating a library to process the roadXML input. Additional enhancements include the generation of NADS miniSim metadata files used in the production of simulator virtual worlds. In previous versions of the converter these files were produced as stand-alone project files, thus precluding the possibility of using the converted design file as part of the NADS Tile Model Library.

Also previous visual model output consisted of simply passing the geometric data through with minimal processing. The resulting model output was a simple wireframe representation of the model (as described in the TIN). This project processes the visual model to generate texture-mapped geometric polygons (surfaces), relying on previously developed texture mapping algorithms from the Safer Sim project **Driving Simulator Use in the Roadway Design and Planning Process**, 2016.

2 Simulation Overview

This section describes in broad terms a high level overview of miniSim simulator architecture as it pertains to the visual and logical representation necessary to support current generation tools and workflow. This overview is helpful to understand which elements the converted model fits into and provides some insight into how compatible the processed converted models are with the standard model library and processes.

The miniSim simulator relies on attribute meta-data to describe and define physical characteristics of roadways, junctions and objects that are presented in the simulation world scene. There are two categories of data: the visual model, consisting of everything visible within the scene, and the virtual model which describes key logical elements of the scene within a correlated virtual environment model (CVED).

These attributes are developed in conjunction with or derive from the visual model to minimize mismatches between the visual and correlated (logical) virtual models.

To facilitate development and production of simulator worlds, NADS relies on a library of component modules that include visual models, model-specific meta-data and a global library of meta-data and attributes referenced by all models within the library. These models take the form of textured 3D geometry and associated meta-data which are collectively referred to as tile models. Tile models must be developed following conventions that define model size, roads, junctions and objects such as signs.

Currently these model attributes are manually generated on a per-model basis, often by re-use of existing meta-data or partially automated through the use of Python and shell command scripts.

Because design models are constructed outside the simulation domain, all potential simulator attributes and datasets within them must be created using information from the input model. At the minimum, these include:

a. Model information: size, objects, roads and intersections (if present), geometry.
b. Road description, including road ID, coordinate data describing the road surface orientation along the length of roadway and connectivity to other roads.
c. Intersection description, including connectivity, surface description (intersection elevation map).
d. Object description: currently objects are not supported by the converter even if present in the design model. Objects are things which have been defined in the scenario object library (vehicles, traffic signs, traffic signals, terrain, etc) and may be re-used across the Tile Model Library. These objects are referenced by one or more models and used in the scenario authoring process, an important part of visualizing environment models (scenery) on the miniSim.

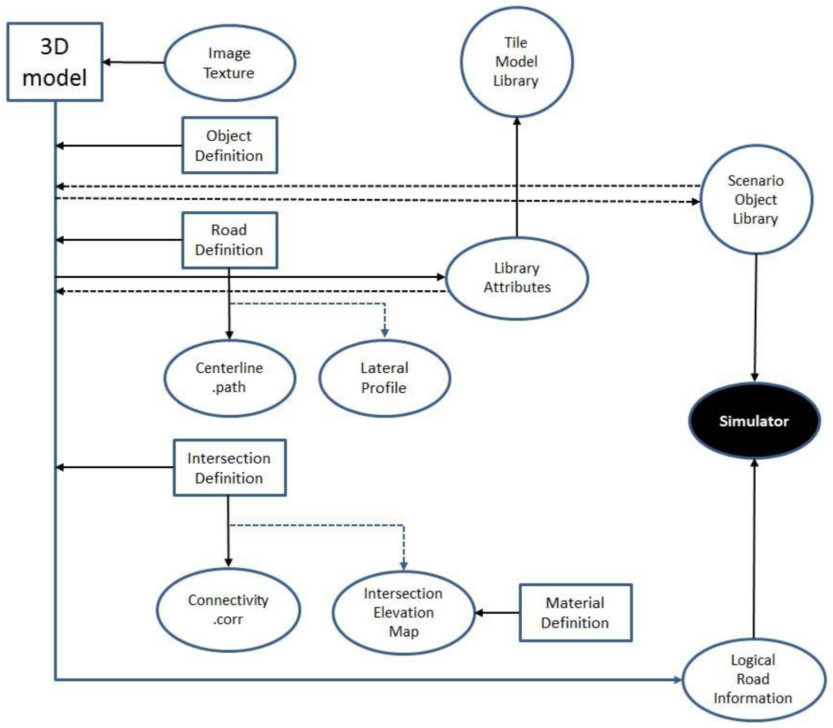

Figure 3 NADS Environment Model Simulator Overview

Lacking road or intersection element definitions in a model that contains the visual representation of these elements would not permit driving on that model, nor allow simulated vehicles to travel on it in the miniSim simulator.

3 Software Enhancements

The converter is designed around a series of processing stages organized into various parsers. This resolves into a parser for each design model attribute from the LandXML file. A number of changes were implemented by this project to improve overall processing and add functionality for the converter. A key improvement involved integrating a library to support XML processing. The original converter was developed around the DOM implementation of the MSXML2 parser from Microsoft. The requirements for a drop-in replacement were: XPath support, written in native C++, speed, simplicity, UTF-8 character encoding, and standards compliance. A number of replacements were considered: Xerces, Expat, XmlLite, tinyxml, RapidXml, and pugixml.

The pugixml library satisfied these requirements best and was available under the MIT license. Making this change reduced processing time from twenty minutes to less than one minute.

3.1 Design model LandXML file format

The design file structure is well documented by AutoDesk™, but in general each design model consists of embedded datasets and transformations to that data as illustrated in Figure 4.

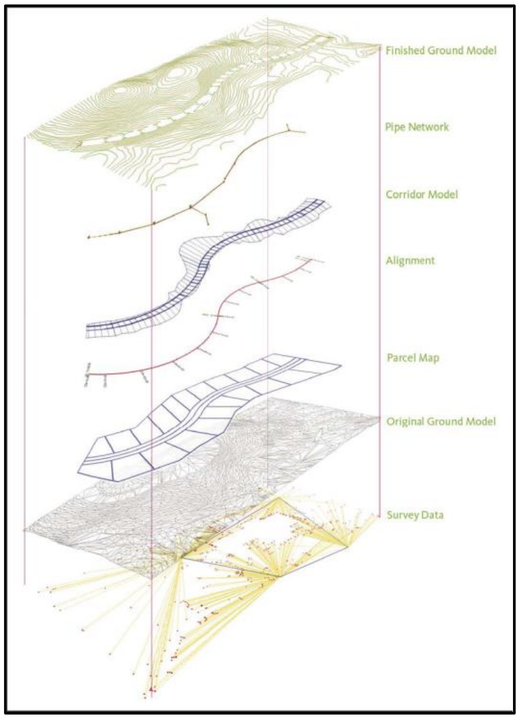

Figure 4 Autodesk Design Model schema (1)

Shown in Figure 4 are a road alignment (centerline), with a corridor model, parcel map and the transformed data in final design form as the top level. The pipe network, ground and survey data layers shown in this diagram are not used by the converter tool at this time because the converter requires the transformed dataset directly pertaining to the roadway and does not process survey inputs.

Each design model must contain attributes needed to support the NADS converter. When these attributes are not found within the design model, the user is prompted to provide them. If these attributes are not provided then further processing is not possible and the converter exits.

A detailed description of the interaction with and converter tool processing of XML data is included in the NADS document ID N12-001 Final Report: Visualization Resources for Iowa State University & Iowa DOT (pp 16-20).

3.2 <u>Determination of roadway geometry</u>

Another performance logjam is the process of extracting road geometry from the rest of the design file, which typically includes significant amount of data that has no direct influence on producing a simulation world as shown in figure 5.

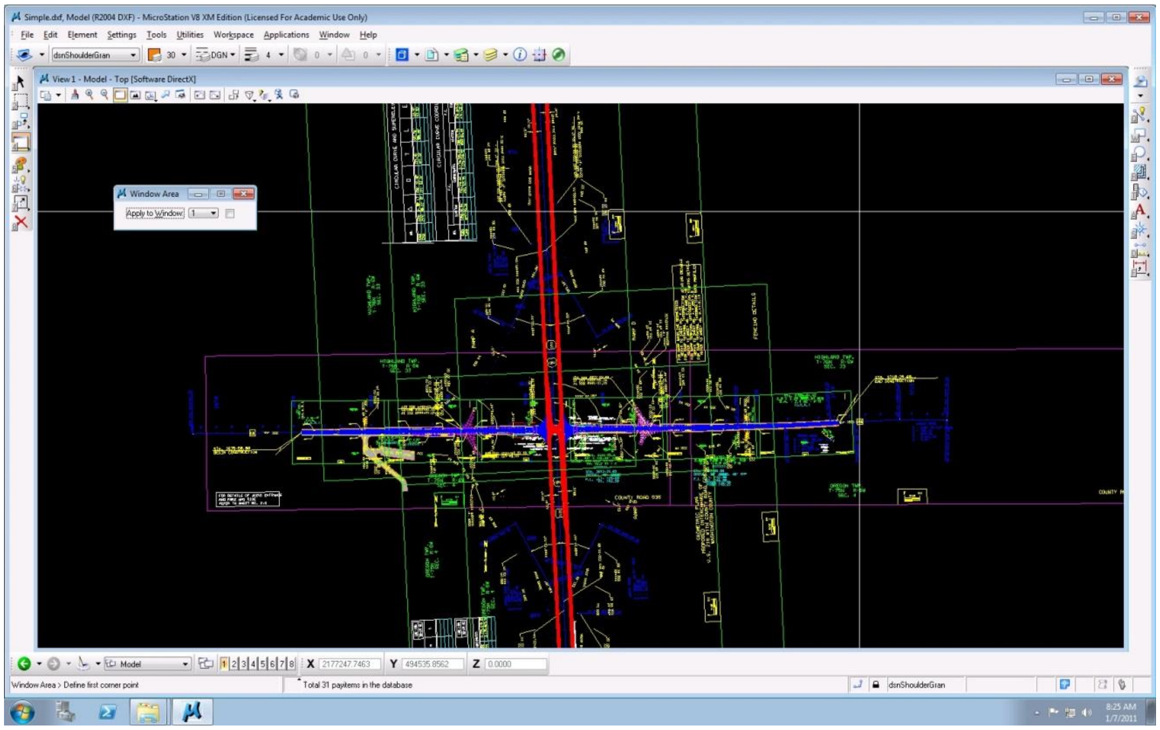
Simple.dxf, Model (R2004 DXF) - MicroStation V8 XM Edition (Licensed For Academic Use Only)
File Edit Element Settings Tools Utilities Workspace Applications Window Help
View 1 - Model - Top [Software DirectX]
Window Area
Apply to Window
Model
Window Area > Define first corner point
Total 31 pay items in the database
dsnShoulderGran

Figure 5 Example Design Model

Earlier versions of the converter tool relied on brute force to determine road geometry: every triangle was compared to every centerline point to see if there was any overlap. This was terribly inefficient, but time concerns kept us from changing algorithms. The typical solution to this sort of problem is to provide some sense of spatial processing, thereby reducing the candidate list, e.g., BSP or quad tree. We were able to achieve significant improvement by including an early check on the distance from the centroid of a given triangle to a given point on the centerline. When this distance was greater than the width of the road, further processing was averted.

With the XML and centroid algorithm changes in place, all of our test cases are processed in less than a minute for each model.

3.3 Intersections

Because of the way driving surfaces are represented in the NADS simulation architecture, it is necessary to divide road networks into roads and intersections. Roads refer specifically to a contiguous driving surface that does not vary in width. The remaining drivable surfaces are considered intersections, and they serve to connect roads together using an explicit *source road*: *source lane* >> *destination road* : *destination lane*: *data* descriptions:

Corridor R2 0 R3 3 CrdrCurves 2ln_4ln_3way_comm_r2l0_r3l3.corr 12.000000

Initially, we sample each design model alignment at regular intervals of around 10 feet. Then, for each pair of sequential sample points, we construct an axis-aligned box that contains these points but is both deeper and wider by a road width (split between corresponding faces). The box also has some height in order to prevent mistakes that might occur when dealing with overpasses. Then, we iterate through all boxes on all pairs of alignments checking for overlap between any two boxes. Any overlapping boxes are considered to be part of an intersection and are marked for further consideration. Next, the boxes are clustered by finding sets with small intervening distances. In particular, the distance between any two boxes must be less than the width of a city block in any direction within the surface plane. The clusters indicate the extents of each intersection.

Then, for each intersection, we find the maximum and minimum coordinates in order to obtain a bounding box for the intersection as a whole. Working backwards everything that is not an intersection must be a road. Given this construct, each road abuts one or more intersections and each intersection abuts one or more roads. By noting how these relate, the connectivity between roads and intersections can be determined.

3.3.1 Intersection corridor smoothing

Corridors are used to link source road lanes to destination road lanes. Defined similar to roads, corridors consist of ordered point data, sorted from origin to destination. Corridor normals are generally ignored; if the intersection contains an elevation map, the surface geometry determines orientation at the corridor point. If the intersection is flat then the normal data resolves to 0,0,1 or vertically oriented.

The converter previously did not support smooth corridors and instead relied on a simple linear here-to-there approach:

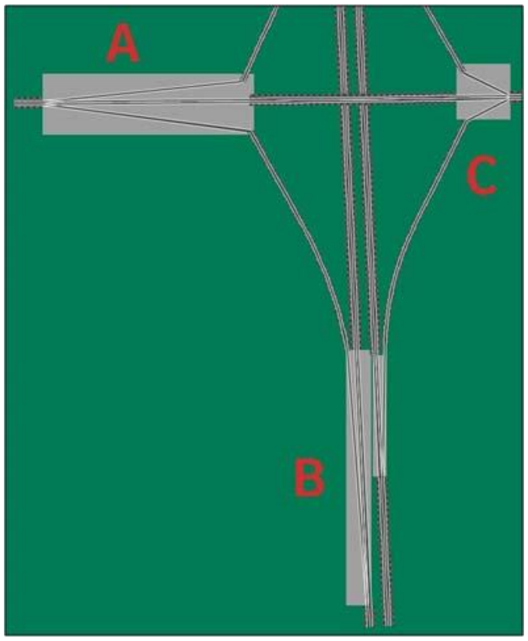

Figure 6 Simple Linear Corridors at A, B and C

While technically valid, the corridors at intersections A, B and C will cause traffic to abruptly change orientation as they travel through the intersection since the transition from road to corridor is not smooth. Abrupt changes at slow speed may be accommodated but at higher speeds traffic may fail when navigating the sharp turns at the intersection boundaries.

The implementation of a spline library allows the converter to produce smoothed corridors. Initially the resulting corridors overlapped adjacent roads. This topology presents a problem for simulator traffic. The disjointed configuration would require vehicles teleport from road to the corridor, which is not valid behavior. This arrangement would cause simulator traffic to vanish as they attempt to calculate a trajectory through the intersection.

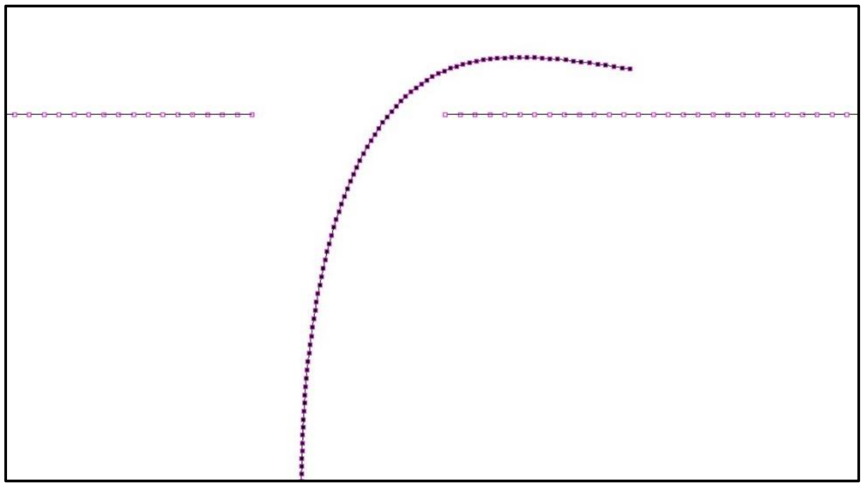

Figure 7 Corridor vs. Road Overlap

The example 4-way intersection in Figure 26 illustrates some remaining tangency and geometric position issues, yet the ADO vehicle can be authored to make a turn and is able to navigate successfully through this region. Larger vehicles are also able to navigate the corridor. However, the largest vehicle model in the NADS fleet, a semi-truck with trailer, is not able to navigate this intersection. This is due to the small size of the intersection and lack of a robust path mechanism for larger vehicles, which require turning wide of the destination lane. These turn trajectories look more like a fishhook than a simple turn as shown by the corridor path below.

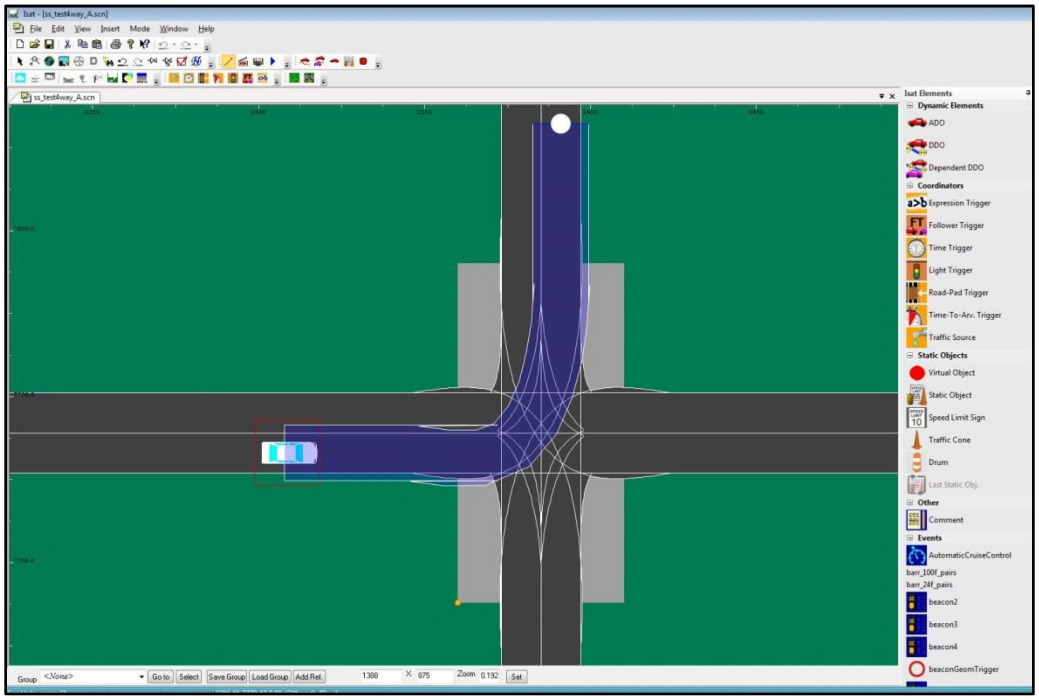

Figure 8 Converted Design Model in ISAT

3.4 Texture Mapping

Texture mapping is the process of assigning imagery to geometry, and is typically a manual operation to enhance the visual appearance of a model. Pixels in the texture map have a unique U, V coordinate ‘address’ that are normalized in the range of 0 to 1.

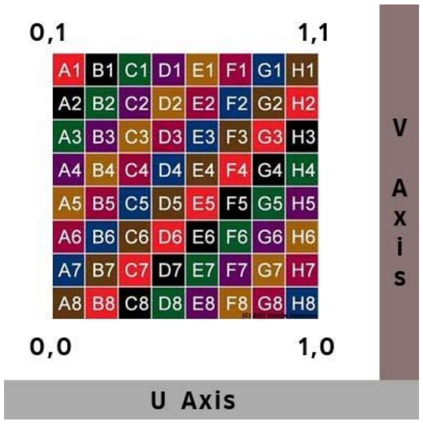

Figure 9 Texture Map Coordinate Space

Texture application becomes more challenging when the appearance of smooth flow is desired on complex surfaces. This is especially true of the region within intersections, where geometric complexity can be considerably higher than roadways as each road splits into multiple smaller ‘ribbons’; visual flow is important and generally requires user direction as illustrated by the UV mapping shown below.

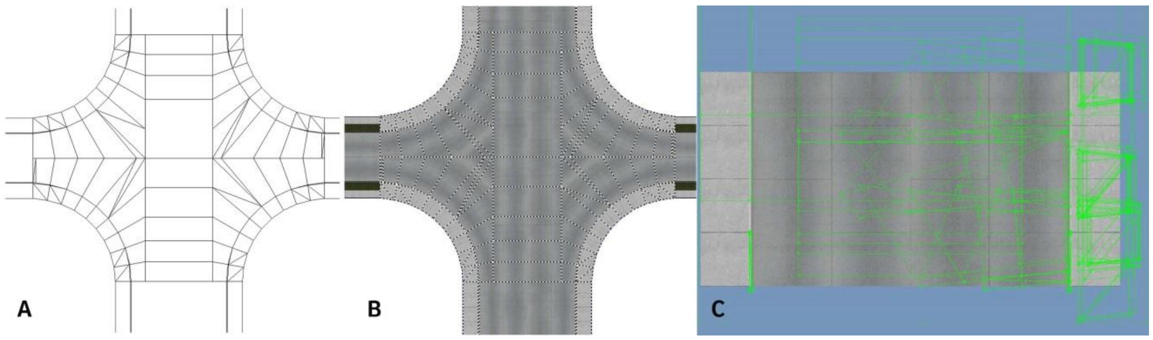

Figure 10 A: geometry, B: manually textured, C: UV mapping

The previous converter did not support texture. The texture algorithm implemented shows some mapping anomalies which means some manual correction is needed to address them in the short term. Texture UVs are skewed and clamped (smeared) as shown in the figure 10 below. The algorithm also completely fails in some regions which is apparent in the output model.

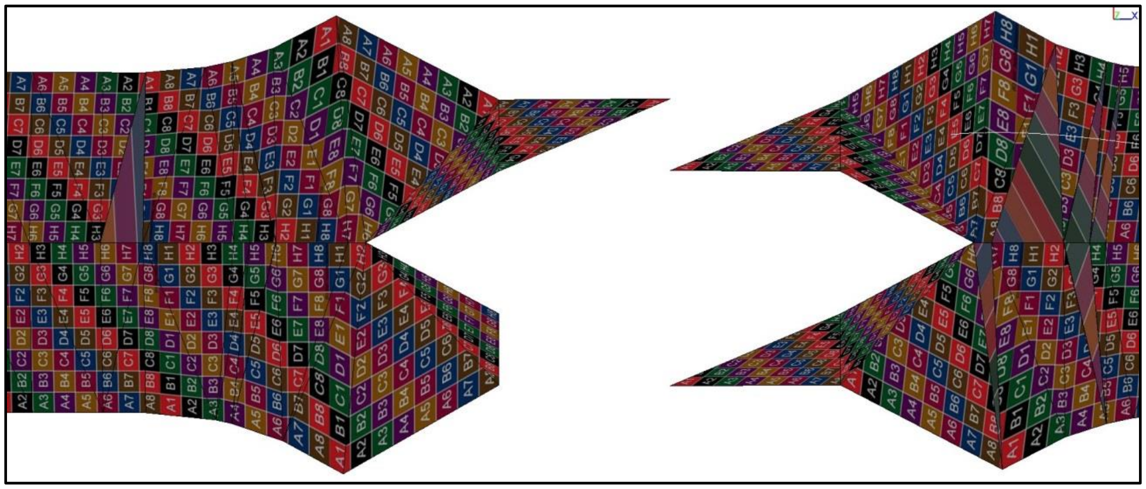

Figure 11 4-way Intersection Region Texture Issues

A potential solution to this issue of complex geometry and algorithm failure would be to implement the intersection as a simplified region and introduce detail in the texture map to emulate the visual effects shown in the manual example in figure 12. This could be achieved nearly irrespective of region complexity through the use of planar projection of an appropriately designed texture. Highly complex surfaces could contain distortions that would likely not be apparent to the simulator driver.

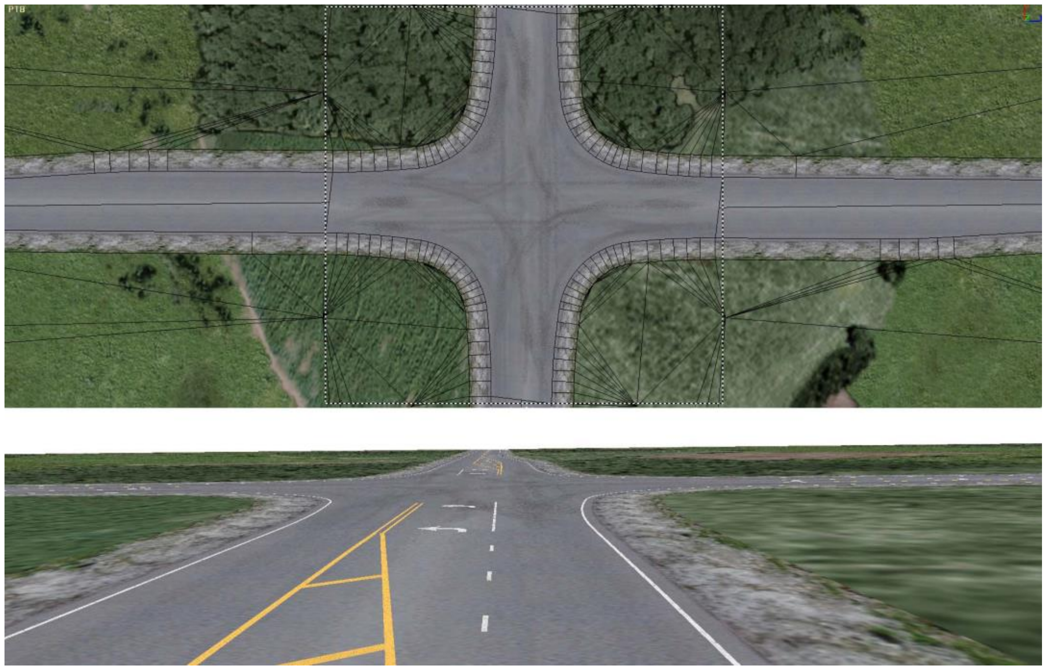

Figure 12 Simplified Intersection Geometry

In the example above, a simple four-sided (quad) polygon has been mapped with a texture designed for use with this particular model road texture and includes appropriate detail indicating traffic wear.

3.4.1 Automated Texture Mapping

The process of automated texture mapping follows the road geometry extraction method described previously that resulted in a pool of triangles. This pool is sorted to reflect the order in which these triangles are located as they progress down the road. The centerline consists of a sorted sequence of points. Segments are defined as the interval between two adjacent points, beginning with the first centerline point and proceeding to the last centerline point. Thus the centerline segment count is always point count – 1.

Triangle sorting is accomplished by projecting each triangle centroid onto each centerline segment, which is a reasonable approach since centerline points (and thus segments) are correctly sorted already.

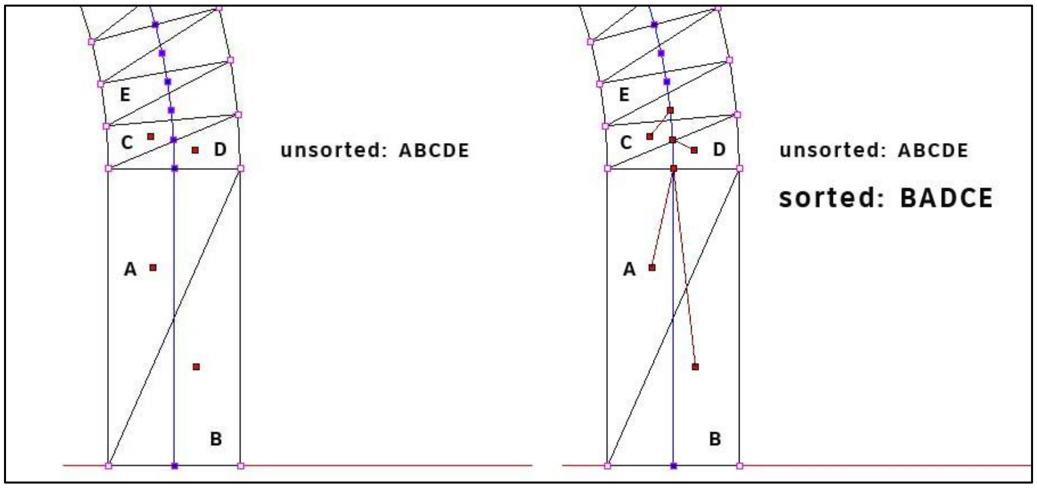

Figure 13 Unsorted and Sorted Triangle Order

If the projected point falls outside the bounds of a segment, we proceed to the next segment. Several segments may pass this initial test. From this group, we compute the distance between each segment endpoint and the triangle's centroid. The index of the endpoint of the segment closest to the centroid serves as the basis for the descriptor which locates that triangle, as each endpoint has an index associated with it that is sequentially correct.

Additionally, we add a fractional amount to the calculated centroid which indicates where along the segment the centroid's projection falls as a percentage of the total segment length. The resulting number will not necessarily agree with any notion of actual length, but it will increase monotonically as one travels down the road. Finally, these numbers, along with their corresponding triangles, are sorted.

Now, with the correctly sorted list of triangles, we must choose a beginning point. To do this, we consider the first two road triangles and identify the common edge and its associated vertices. The one unrepresented vertex then must be "outer-most" and is selected to be the point of origination for the road. This is typically mapped to the origin in UV space, but may be the point (0, 1) if it is determined that it is on the left side of the road. In our implementation the U axis lies along the road's longitudinal axis and the V axis extends laterally across the road.

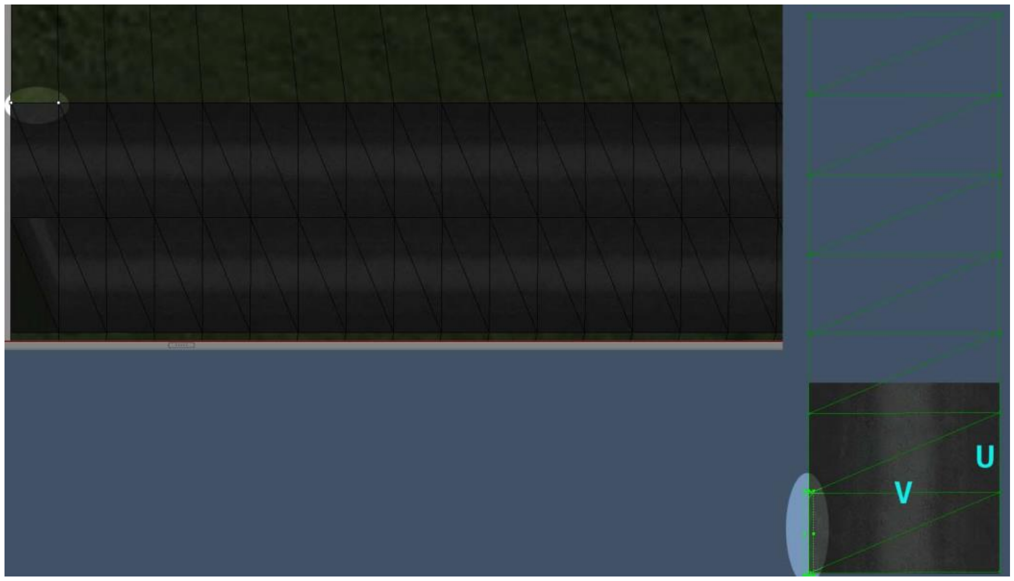

Figure 14 Texture UV assignment

The triangle must be transformed in UV coordinate space such that it aligns with adjacent triangles in order to avoid introducing texture anomalies (such as skewing or stretching) that are evident when the texture is applied regardless of 3D orientation. Before this can be done, however, any offset that accumulates as we move along the list of triangles must be accommodated. This method ensures that the triangle rotates about the triangle origin, as opposed to revolving around the model origin since that would produce invalid results for consecutive (flowing) texture mapping.

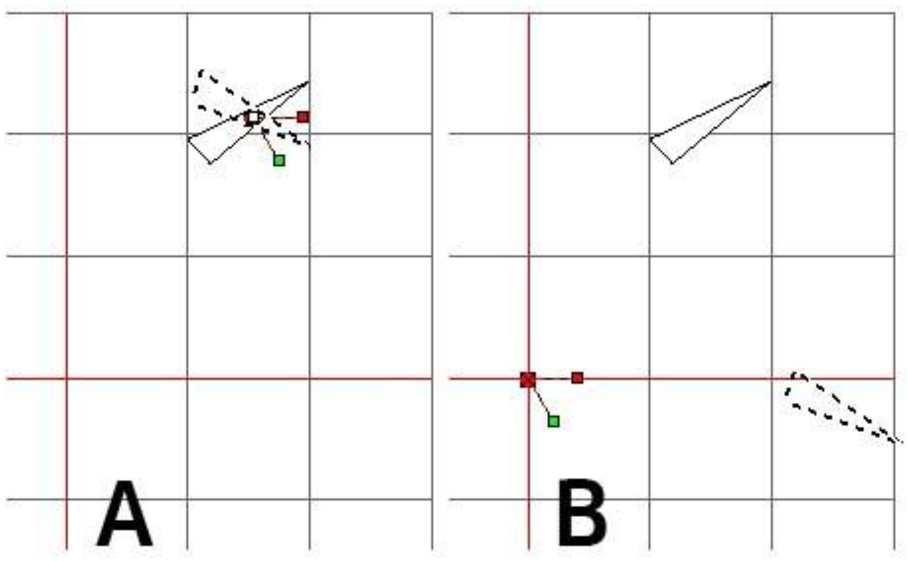

Figure 15 Triangle Origin Rotation vs. Model Origin Rotation

In order to perform these transforms, the necessary Euler angles are computed with basic trigonometry. Then a standard transformation is applied to each vertex to do the rotation. This same process is repeated for the remaining triangle adjacent to the first centerline segment. Since we know the triangles geometric length, we can scale the triangle appropriately such that a unit in UV space corresponds to the width of the road.

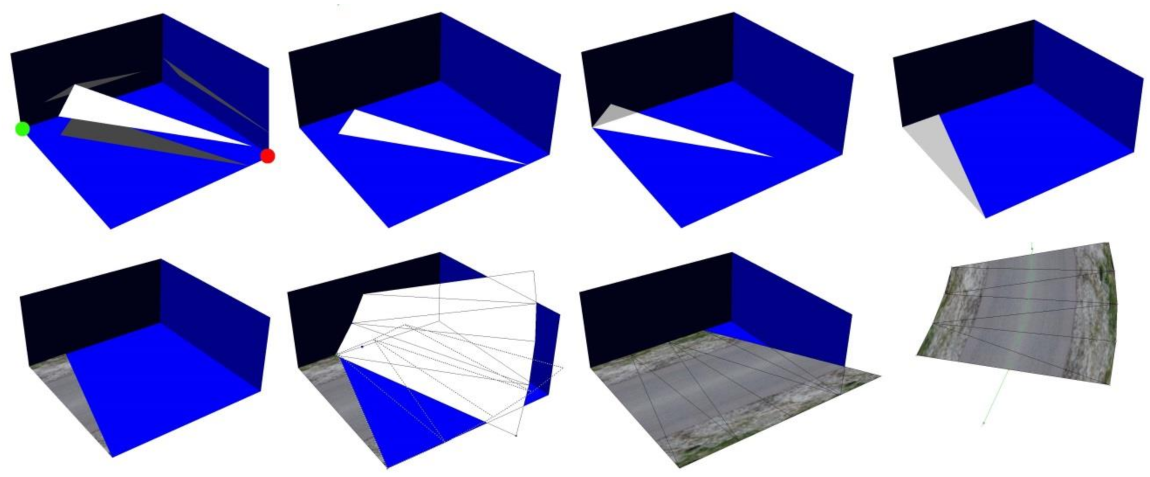

Figure 16 Transformations to Assign UV Coordinates

Now, we can generate two new lines that are offset from the centerline by half a unit that represent the left and right sides of the road. Using these as references, we can determine whether a vertex falls on the left or right edge of the road. All rightward vertices have a V texture coordinate of zero, and all leftward vertices have a V texture coordinate of one. The U coordinates can be obtained by a rotation within the UV plane that aligns the points with the V = 0 or V = 1 line.

This technique does not seem to work well for curvy roads or for roads that vary in width as implemented. There is clearly room for improvement in this algorithm.

2 Test Cases

Three initial test cases consisted of simple roadway ribbons, containing road geometry that was processed to determine the feasibility of extracting road centerline data from an unordered mass of geometry. One file included terrain geometry as shown in figure 10 which proved to be problematic and was later removed for development purposes.

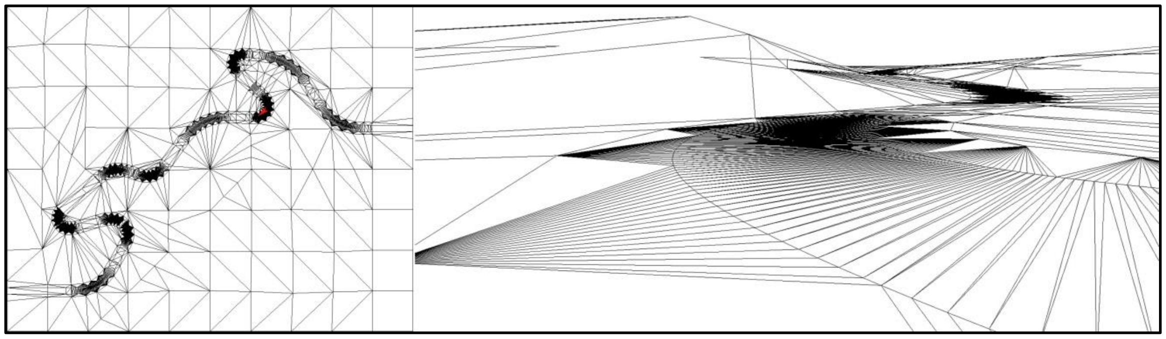

Figure 17 Hilly, Winding Road Test Case

A hilly, winding road was considered a good test case for the texture application process because this type of geometry is significantly more complex than straight or simple hills when calculating sequential UV mapping. A more extreme example was also tested with elevation changes over 300 feet.

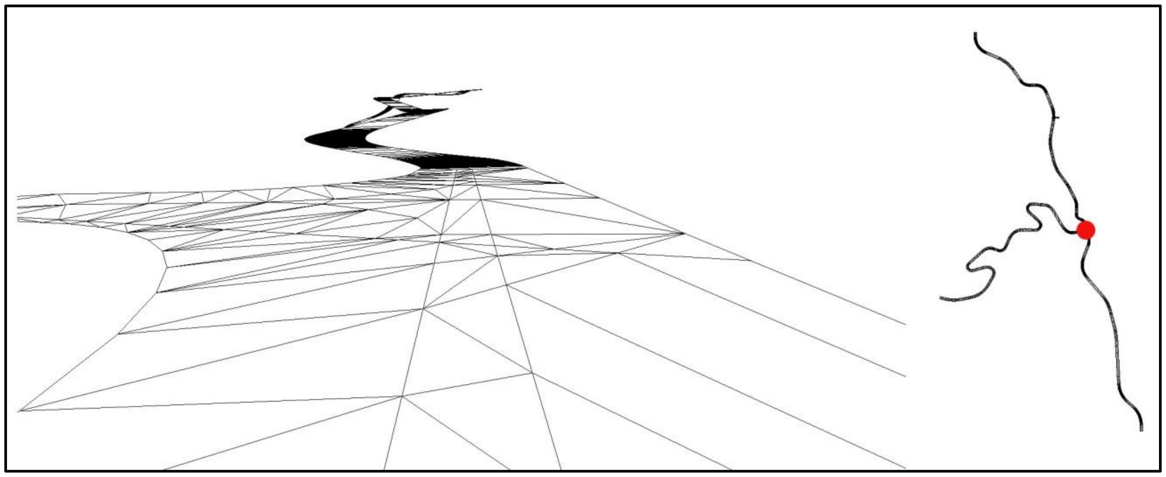

Figure 18 Mountain Road Test Case

In addition to these simple geometric test cases, additional files were created in AutoDesk Civil3D as simple roadway design models to cover a broad range of design types, ranging from simple straight roadway to a more complicated road network. These test cases consist of the minimum design information needed to support the converter and do not include real-world design model datasets such as a site plan or survey data. The test models contain a majority of the design considerations which are used in the miniSim simulated worlds, including super-elevation, rising curves, overpasses, a road split commonly found in freeway on/off-ramps, intersections and acute angle intersections. Test cases also included one lane roads, as is typical of some interchange ramps.

- Straight Roadway (simplest, smallest test)

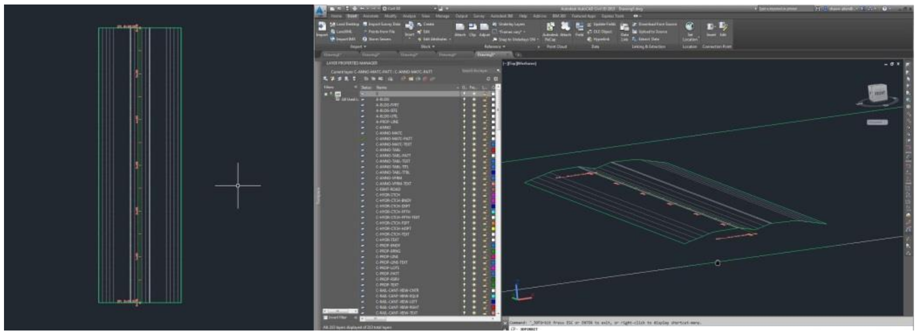

Figure 19 Straight Road Test Case

- Curve

- Hill (rising curve)

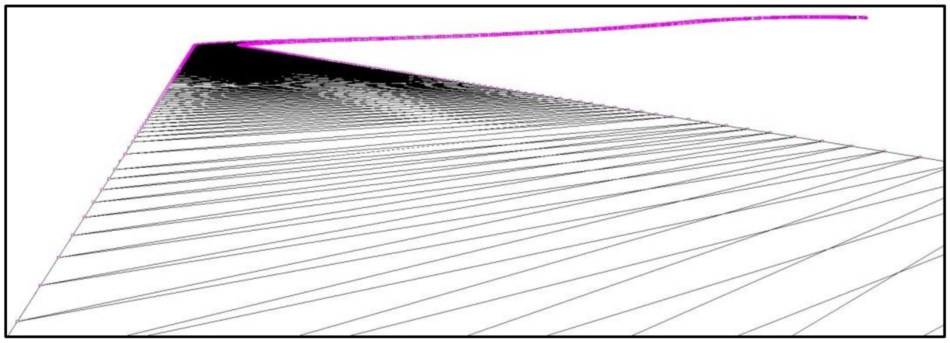

Figure 20 Combined Curve and Hill Test Case

- Boulevard
- Divided Highway

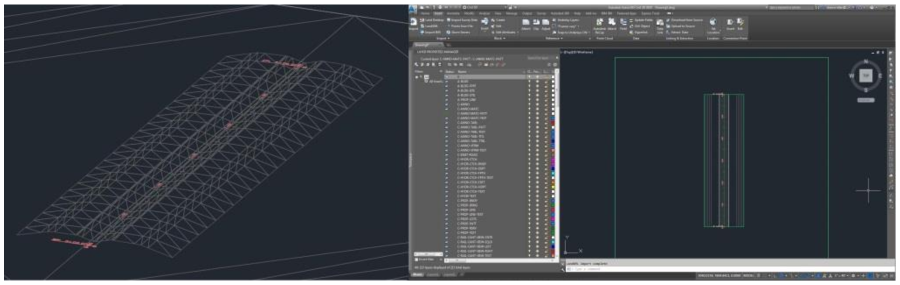

Figure 21 Divided Highway Test Case

- Roadway with barrier rail
- Simple Overpass
- 3-way intersection

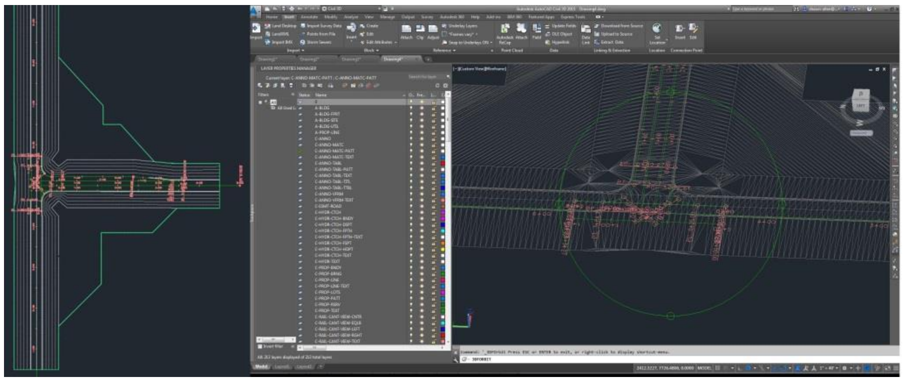

Figure 22 3-way Intersection Test Case

- 4-way intersection

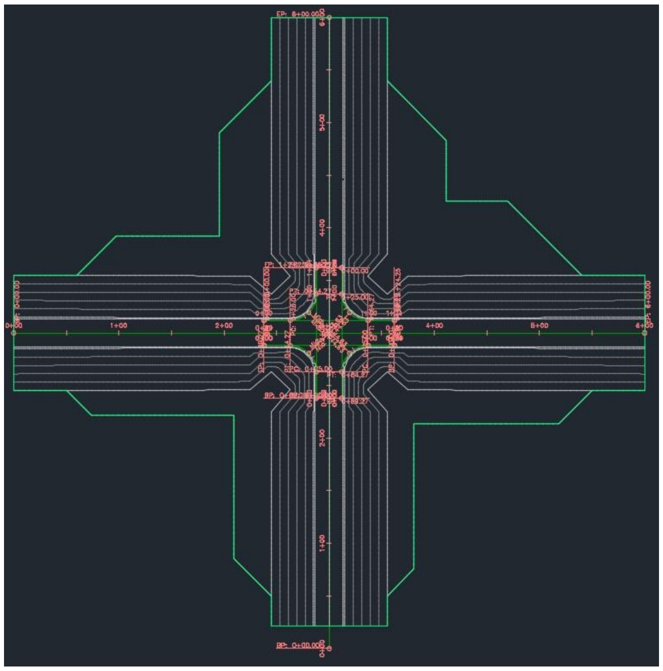

Figure 23 4-way Test Case

- Freeway on-ramp

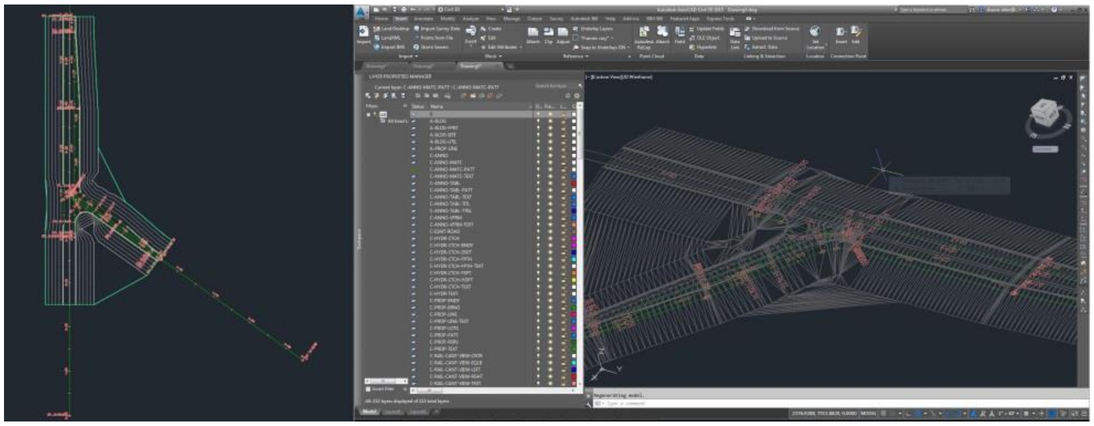

Figure 24 On-ramp Test Case

- Roundabout

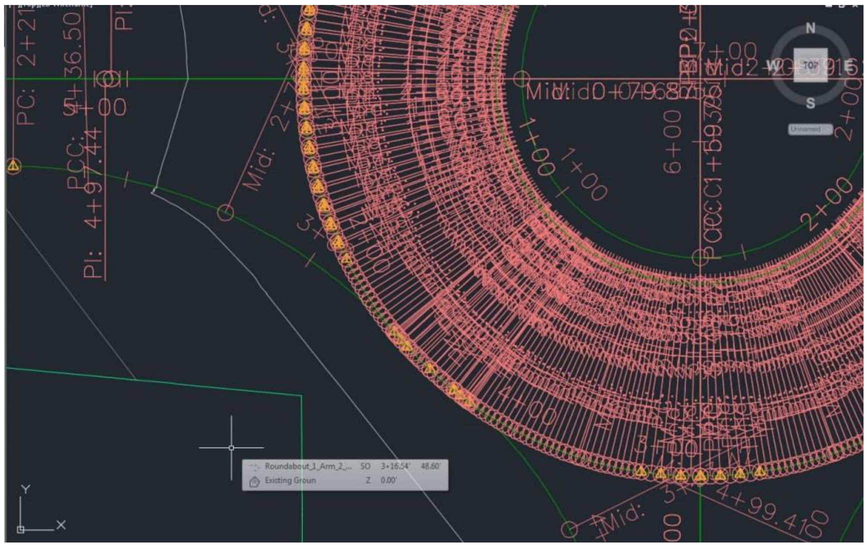

Figure 25 Roundabout Test Case

- ‘City’

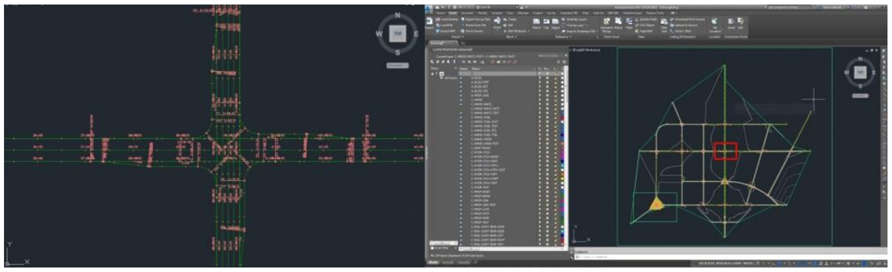

Figure 26 ‘City’ Test Case

3 Workflow Overview

Workflow is the work and procedures needed to convert a design model and visualize it on the NADS miniSim. This process currently involves running the Design Converter to produce simulator model files, integrating those files into the Tile Model Library, creating a scenario world that uses the converted file, then creating a scenario to drive and installing the world on a miniSim.

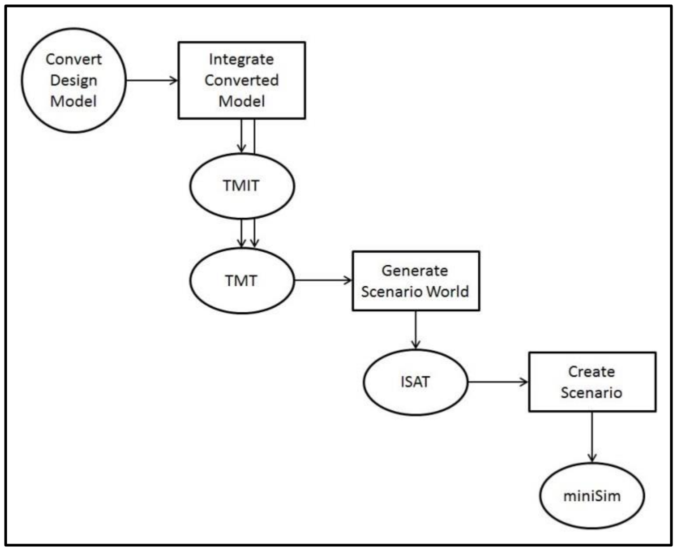

Figure 27 Conversion Workflow

After the design model has been integrated into the Tile Model Library, standard build procedures must be used to generate the visual and data files necessary to author a scenario and visualize it on the NADS miniSim. This process generates multiple visual files (geometry), texture and road network data in the form of a BLI file. The BLI file is used as a map for the NADS Interactive Scenario Authoring Tool (ISAT) to create scenarios (interactive visualizations) on the miniSim. Installing the generated files and importing the scenario into miniSim allows the user to then drive their converted design file.

4 Converted Model Integration Tasks

In order to use the converted design model with the NADS miniSim, a number of integration and build tasks must be completed. The majority of these integration steps are conducted using a graphical user interface to the Tile Model Library called the Tile Model Integrator Tool (TMIT). Both library parameter and tile specific files must be updated or generated. Build tasks are conducted using standard methods which are described in miniSim training(2) and Tile Mosaic Tool Manual(33).

4.1 Tile Model Library

The Tile Model Library is a collection of files and folders known as the Tile Model Library, or library. These files include parameter files for the collective model set and geometry and data files for each individual tile model in the library. At this time, the directory structure and system environment variables are also key elements of the library. Therefore in order to use design model files on the miniSim, the converted design model files must be integrated within this framework of folders and files.

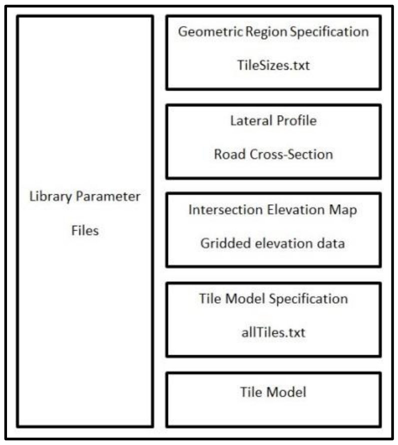

Figure 28 Tile Model Library schema

The interface to the Tile Model Library is through a graphical 2D map editor called the Tile Mosaic Tool^{TM} (TMT). Users place tile models onto a map and then generate visual and logical data for each configuration (world) needed. Files that have not been integrated into the library are not visible to the TMT even if they reside within the library folder structure.

Tile models are contained within model folders organized into category folders. Each tile model name must be unique and includes various data and meta-data files that define the model and roads, intersections and objects within the tile model.

Each tile model also requires specification for each road and intersection contained in the tile model within meta-data records and data files. These records follow NADS LRI conventions as documented in the NADS LRI Specification document. According to convention, roads are ribbons defined with a centerline and surface orientation; this data is stored in a file specific to each road. A single tile may contain one or more roads. The example road definition below shows a 2 lane city road with a default speed limit of 55 miles per hour. This speed attribute is used by autonomous traffic during simulation.

RoadDef RoadName r1 **RdwayCrv** test_r1.path

LaneDef 12.000000 P

LaneDef 12.000000 N

DefRdAttr SpeedLimits 55.000000 0.000000

DefRdAttr CityRoad -1 0.000000

The above road definition example shows required keywords bolded and parametric fields highlighted. The data for each road centerline is contained within the file specified within this definition, test_r1.path.

In addition to basic road information, an optional lateral profile describes a cross section for each road. Lateral profiles describe the shape of the cross section and surface material across the road. These attributes support vehicle dynamics by associating materials to a coefficient of friction value. Each change in the road surface for a road requires a unique lateral profile as shown below:

DefLatCrvFile R1 **LatProfileList.lat LatCrvName** RES_LN2 For example, if this particular roadway contained a material change (asphalt to concrete) then it would require a lateral profile for each material used.

Lateral profile data is currently contained within the file LatProfileList.lat and referenced by the ID in the last record field in the above example. This lateral profile file is referenced by all tile models in the library.

Models containing multiple roads require the use of connectors or junctions called intersections to join multiple road segments into a road network. Intersections may be flat, elevated or sloped depending on the nature of the road surfaces at the junction location. Flat or elevated intersections may be defined within a tile intersection definition, but sloped, articulated or complex intersections must use an elevation map to describe the surface. This data is contained within the Intersection.map library file and is commonly unique to each model. As a result, this file has become too large to be manually edited and requires map data to be added into intermediate files, which are then concatenated into the parameter file used during the LRI generation process.

4.2 Tile Model Integrator Tool

The TMIT performs processing steps that would otherwise be tedious and error-prone on library parameter files as well as tile model files. Written in Python, TMIT can be extended by any knowledgeable Python programmer. TMIT manages the tile model by copying it from a general location to a Tile Library folder, converts between file geometry formats and prompts the user to enter required tile model parameters.

OpenFlight^{TM} is required for the TMT and is a well-documented simulation file format created, supported and maintained by Presagis^{TM}. Since the converter generates Wavefront^{TM} OBJ files, an additional conversion (to OpenFLight) is needed. This conversion is managed by the TMIT tool after selection of the model to process. The actual conversion is performed by **osgconv.exe**, which generates a .FLT file. The use of a 3rd party geometry converter provides leverage to use additional input file formats without the responsibility of developing converters for individual file formats.

Another required tile model file is the preview icon which is used by TMT for each model. The file format is an 8 bit AT&T .icn file which is generated by the 3D modeling software Presagis^{TM} Creator^{TM}. Since this tool is generally not available to miniSim users and the file format is not well documented, a generic icon may be used in place of the tile model icon. This approach means there may be no visual clue to the model appearance within the TMT. A generic resource .icn file is included with the TMIT to automatically generate this file.

5 Converter Tool Use

This version of the converter remains a command line tool where it is executed from a DOS shell:

XMLConvert <LandXML input file> <output prefix> <Enter>

Several output files are generated into the same folder as the input file, assuming the command is generated from the same location as the input:

Table 1 Converter output files

File name & type	Description
{output prefix}.PET	one per design, contains meta-data definitions, references to library elements and connectivity information
{output prefix}.ISXN	Intersection record data, one per intersection
{output prefix}.CORR	one per corridor
{output prefix}.MAP	elevation map, one per intersection
{output prefix}.ROAD	one per road segment, per LRI specification
{output prefix}.LAT	lateral profile, one for entire design
{output prefix}.OBJ	visual model of road network from input design model
{output prefix}.MTL	default material for the visual model
{output prefix}.TOP	legacy file currently not used

These output files must be integrated into the miniSim workflow before the model is available for use on the miniSim. TMIT provides a graphical user interface to integrate these various files into the Tile Model Library. This tool is well documented in the SaferSim report **Driving Simulator Use in the Roadway Design and Planning Process** (pp. 34-53).

6 Conclusions and Future Work

Several problems remain when processing intersection corridors. The selection of points used to construct the spline was a bit too cavalier, and, as such, the shape of corridors does not match well to circular arcs. In particular, the paths do not have enough curvature at the start which means there is too much at the end. Computer-controlled simulator vehicles can negotiate these paths, but they expose a feedback loop in the control logic that cause vehicles to lose control further down the road shortly after navigating the corridor. Changing the vehicle dynamics sampling frequency can help (or hurt), but this would require case-by-case analysis. Corridors overrun intersection in some left turns. Furthermore, corridors and roads do not necessarily meet and rarely meet at intersection boundaries as shown in Figure 22.

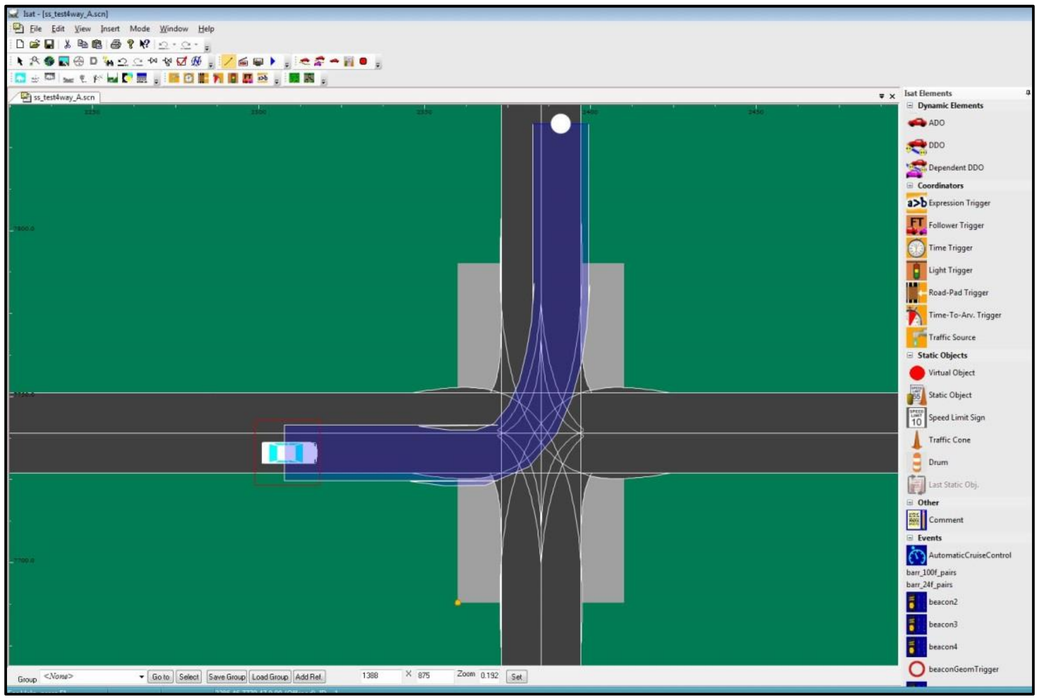

Figure 29 Corridor, road boundary overlap

A more rigorous solution will ensure the road direction vector is maintained into the intersection and not overlap the road.

Due to software limitations, a number of changes must be made to the input design file. For each road, there should be only one alignment, namely the centerline. Side-profiles are required for all alignments. Pave1 and Pave2 layers, (or comparable layers) which represent the road surface and shoulder, must be included for at least one cross section per alignment in order to indicate the left and right road edges.

Processing model geometry present in the TIN records currently also exposes a weakness of the converter. There may be only a single TIN surface within the input file, and that should correspond to the finished road. Processing time can be influenced significantly by the number of triangles within the TIN, so it may be desirable to cut large designs into pieces.

Manual intervention may be needed during processing to select the correct alignment or surface when more than one reasonable selection exists.

Additionally, roads with more than two lanes of traffic frequently end up with texture mapping errors.

Complex designs as shown below, including traffic circles or roundabouts remain uninterpretable by the converter.

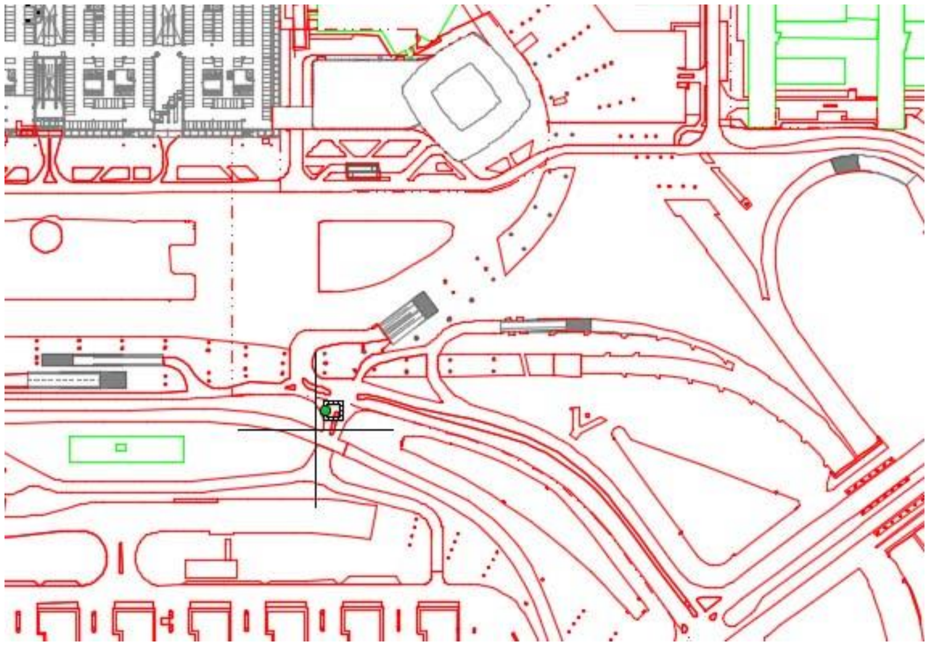

Figure 30 Complex road network example

As mentioned previously, additional work is required to fully integrate the converted model files into the miniSim workflow. Elevation maps are represented in a single Tile

Model Library resource file. Therefore, the converted model .MAP file must be integrated into this resource. The same is true for lateral profiles.

Numbers occasionally need to be truncated in order to be parsed correctly due to the imprecision of floating point coordinates with 12-16 or more decimal places.

Additionally, XMLConvert does not currently extract road attributes (e.g., default road speed limit or bike path tags) from the design files. However, at least one attribute is required to satisfy the miniSim world build process, so one must be added manually.

The geometric processor must be enhanced to support larger TIN files, including non-road geometry. It would also be useful to create generic terrain in the output model instead of omitting it entirely.

Lastly, the converted models maintain their original road design coordinates. In order to be fully compatible with the NADS Tile Model Library, these coordinates should be transformed such that the model origin (0,0,0 point) rests at the lower left-hand corner of the tile per modelling conventions.

Glossary

Alignment: design model centerline

Assembly: design model cross section of a road

Centerline: the center of a road

Corridor: simulator model data structure and file to define road network connectivity

CVED: correlated virtual environment database; data description of visual model and characteristics (speed limits, virtual actors, etc)

DOS: Disk Operating System; often used to describe the command line PC interface where commands are typed in and executed

DWG: Autodesk design model file format

Elevation Map: elevation data structure defined as a column/row grid containing elevation and material type

Intersection: simulator model junction that associates adjacent roads

ISAT: Interactive Scenario Authoring Tool; the graphic user interface and workspace to create events or tune the simulated environment

ISXN: see intersection

LandXML: design model file format in text form, processed as the primary input to the nads converter tool

Lateral Profile: Simulator model cross section of road that describes the elevation and surface materials across the road; a TMT library asset

Layer: design model construction or management hierarchy element

Metadata: information that provides information about other data [wikipedia;https://en.wikipedia.org/wiki/Metadata]

miniSim: the NADS portable driving simulator

MTL: Wavefront texture/materials file, associated with OBJ

Normal: geometry reference direction that is perpendicular to some reference point

OBJ: Wavefront geometry model file, associated with MTL file

PET: Polytag Extraction Tool file that contains road, intersection and object definitions

Road network: two or more simulator model roads

Road: simulator model includes visual geometry and data that describes center line and surface attributes

Scenario: A series of actions designed to present the simulator driver with one or more tasks, or to present situations where the driver responses are of interest

Segment: edge between two adjacent centerline points

SOL: Scenario Object Library, containing objects used in the creation of simulation scenarios and used as features in tile models

Super-elevation: horizontal cross-slope of a road; used as design parameter to permit vehicles to traverse curves at speeds greater than would be possible if the curve is flat

Terrain: simulator model visual representation of non-road, non-feature surfaces

Texture: bitmap file applied to geometry to create a visually compelling model

Tile: simulator model consisting of visual geometry, texture and data files to describe roads, intersections and road connectivity, road type and speed limit attributes and other scenario objects

TIN: triangular irregular network; a data structure for the representation of one or more surfaces, usually terrain

TMT: Tile Mosaic Tool, a graphic user interface and workspace to arrange tile models to construct simulator scenario worlds

UV: texture mapping domain that supports assigning texture to model geometry

Vector: one dimensional (directional) array; contained in road data files to describe the road surface orientation at each centerline point

Workflow: a sequence of actions or processes that produce a desired resultReferences

1. Crews, Nathan (2007). Autodesk Civil 3D 2007 LandXML Support.doc
2. TMT training video at: http://www.screencast.com/t/KubOfsQzHGh6
3. Tile Mosaic Tool Manual, Sections: Creating Output, Output for Visual Database, Output for Correlated (Logical) Database

Appendix A: NADS LRI Specification

The NADS LRI Specification document is available from NADS as a separate document.