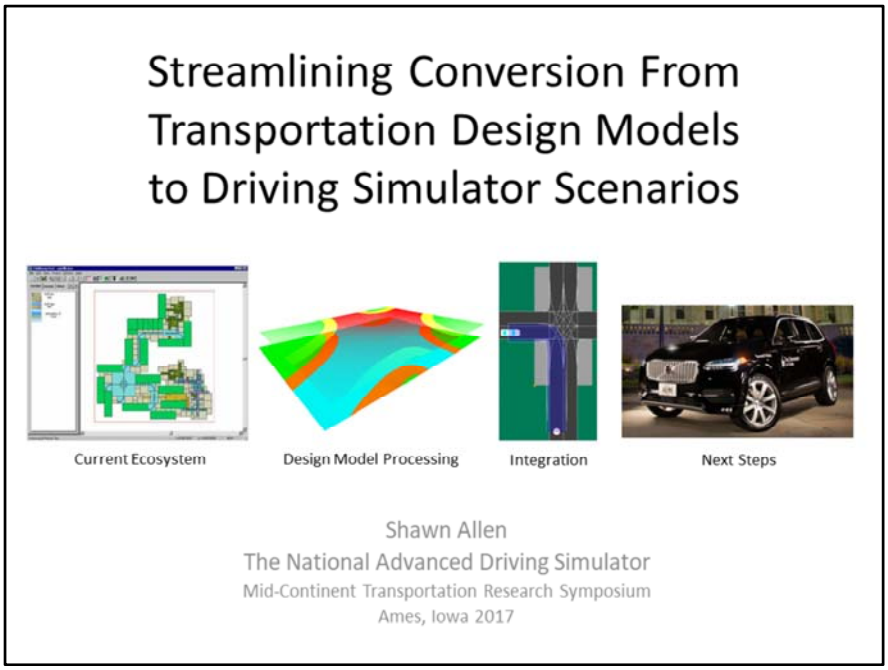


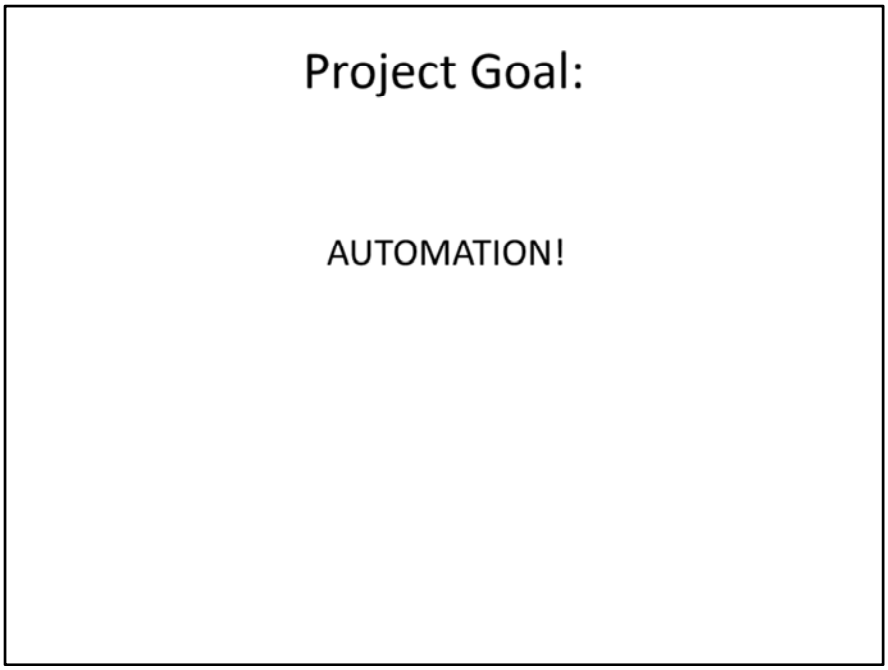
The goal of this project is to convert design models to NADS driving simulator scenarios. The information I'm presenting here today is not needed to use the converter, but understanding the bigger picture puts some of the design details of the converter into context.

I'll be presenting an overview of the current NADS simulator ecosystem to show how the virtual modeling aspects have been implemented historically.

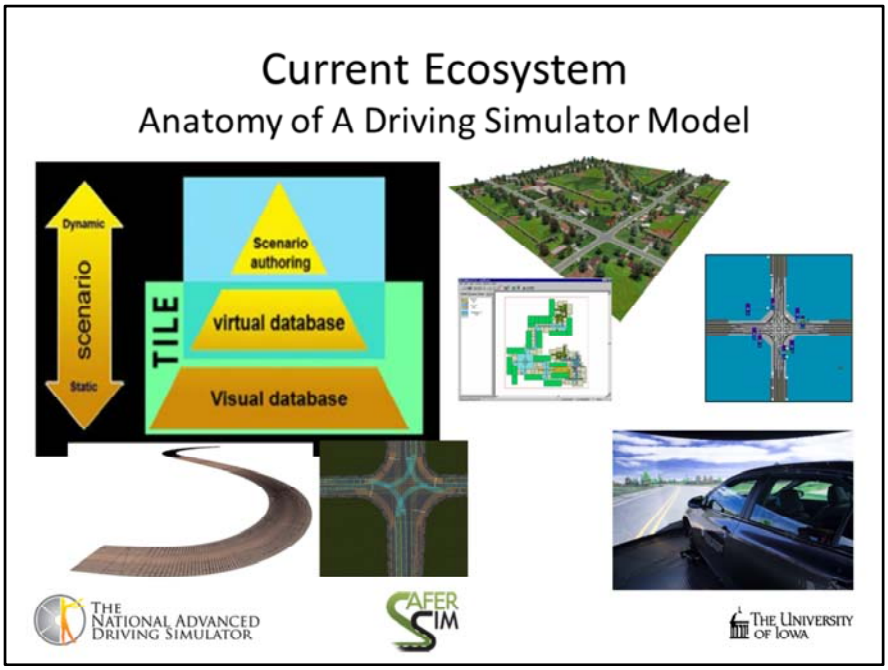
I'll also discuss what processing steps happen to the design file, and how this version of the converter is moving closer to integrating with historical simulation assets that have been built prior to converting.

Since the converter now generates numerous files per design, managing the integration of the generated output is an important and necessary step.

Finally I'll cover some of the limitations that were uncovered and what the next steps are.



More specifically this project is intended to automate the conversion and integration of the converted files into the NADS simulator architecture. This process should be possible by anyone with a working knowledge of windows applications.



Nads simulation is based on the notion of two types of data that are connected, or correlated to each other. These concepts apply to both individual models and the overall scene, which is built out of these individual models.

At the bottom of the model pyramid is a visual model or database. This is the scenery visible when driving the simulator. This scenery consists of simple geometry and image texture maps to provide detail.

The second type of data which is separate from the visual model is the virtual or logical database. This consists of data and attributes needed to define different things in the simulation, for example, defining what a road is and how it looks to vehicle dynamics in terms of surface material. This is also where a road network is defined. Portions of this data are generic and apply to the collection of models as a whole.

These two types of data: visual and virtual collectively define a simulator model.

As you can imagine, roads are a significant element in a driving simulator. Roads are defined with a centerline, which is a series of point data sorted to proceed from one end of a road to the other. That means a road also has a direction – it travels from

some place to some other place.

Roads are also limited by convention to be fixed in terms of the number of lanes, and they connect to other roads through intersection junctions by means of an explicit declaration of connectivity called corridors. Adding or dropping lanes must also happen through intersections.

This is an example of a moderately complex tile model. It's a residential tile containing 4 intersections and visual features consistent with a school zone. If you wanted you could create an entire city from this one tile – although people are good at detecting patterns.

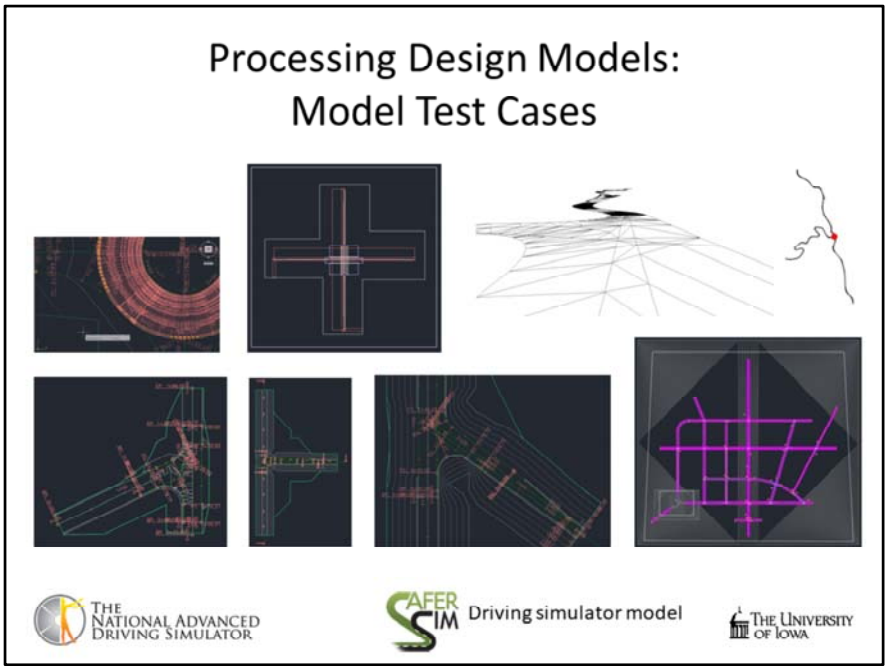
Now we come to the layout tool for creating simulation worlds. Tile models can be copied or rotated and placed to create different environments. After the layout is finished this tool creates the visual and virtual databases. The virtual database is then read by the interactive scenario authoring tool where events are created for interactive simulations.

The 4way intersection shown here is a traffic controlled intersection. After creating a scenario, it's then possible to drive it in the simulator.

The simulator user generally deals with the right side of this slide – what the scene should look like, how long should the drive be, and what does the driver experience during simulation.

The concepts and details on the left side are managed by model construction and tools.

This is the overall framework that the converter has been designed to work with.



This slide contains most of the models we tested the converter with. Our student Adam used Autodesk Civil3D to create simple design models for various road and intersection types. The goal here was to create a variety of roadway types, including single and multiple roadbeds, flat and hilly, etc. One of our test cases included a barrier feature.

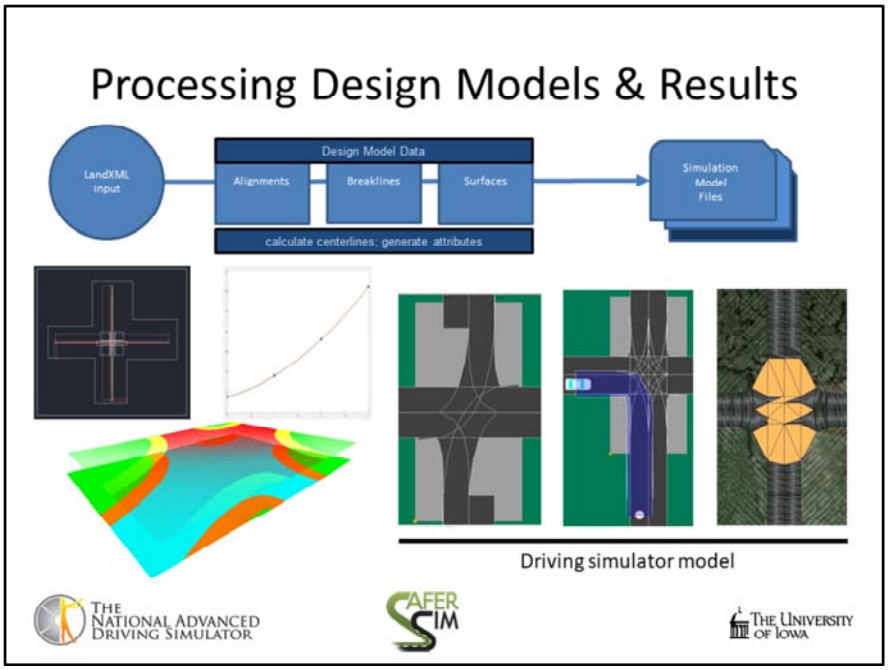
Our starting point was part of an existing tile model. The goal here was to see if the converter was able to extract centerline data from an unsorted pile of triangle geometry. The initial test included terrain, but this proved to be a problem and so the road surfaces were extracted to simply the problem.

Several flavors of 3way intersection were built for various types of possible roadway configurations. The angled junction could also be considered a merge or freeway ramp.

The 4way intersection is a simple flat and level intersection.

The traffic circle breaks the converter, probably due to the algorithms that split roads into sections and defines intersection regions.
The last test model was a simplified city containing all the other test cases.

All of these test models are processed in about a minute, which seems to be an indicator the algorithms are somewhat scale neutral.



There is an assumption with this version that each design file is a self-contained simulation model, or tile.

Some of the enhancements made to this version of the converter included the integration of an XML library, which is where some of the improvement in model processing speed comes from.

The converter first tries to determine road centerlines from the design model alignments in the landXML exported from Civil3D. In cases where the converter is unable to determine a centerline, the user is prompted to select it. If no centerline can be determined then the converter will exit.

The road network is then partitioned to determine where intersection junctions are needed. Some of the intersection attributes are hard-coded into the tool for the moment. For example, half a road width is considered a minimum intersection dimension.

After determining the intersection area, the converter then examines the surfaces under the intersection region and generates an intersection elevation map file for the

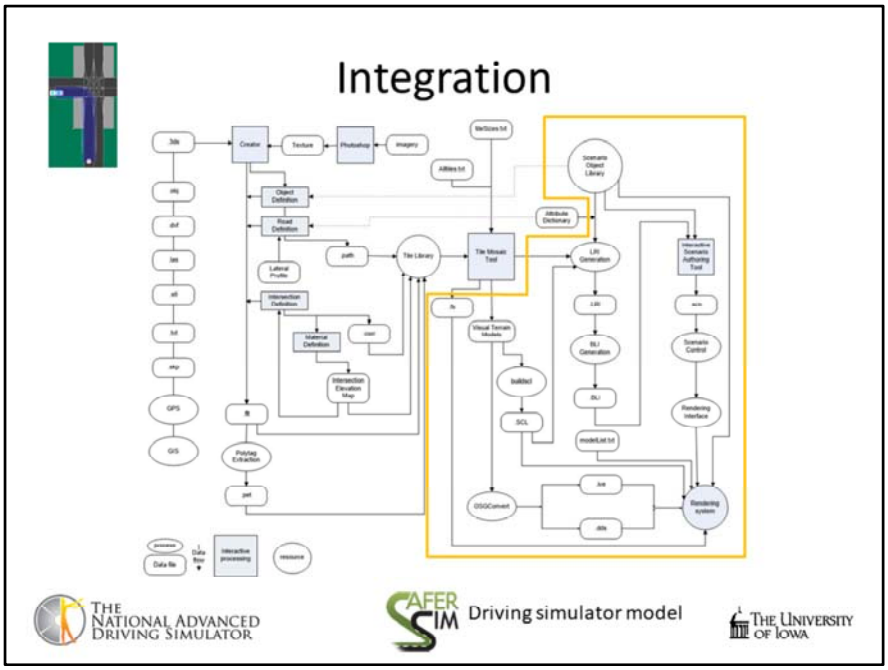
design model.

Currently all possible connections are considered valid, and these connections are simple arcs generated from a spline library. This is a tool view showing the virtual database for the 4way intersection model after conversion.

There are some interesting problems that have shown up due to the way intersections and roads are segmented. In the case where a road/connector boundary is not clean, simulated traffic can go into a feedback loop and crash unless they are traveling at a low rate of speed.

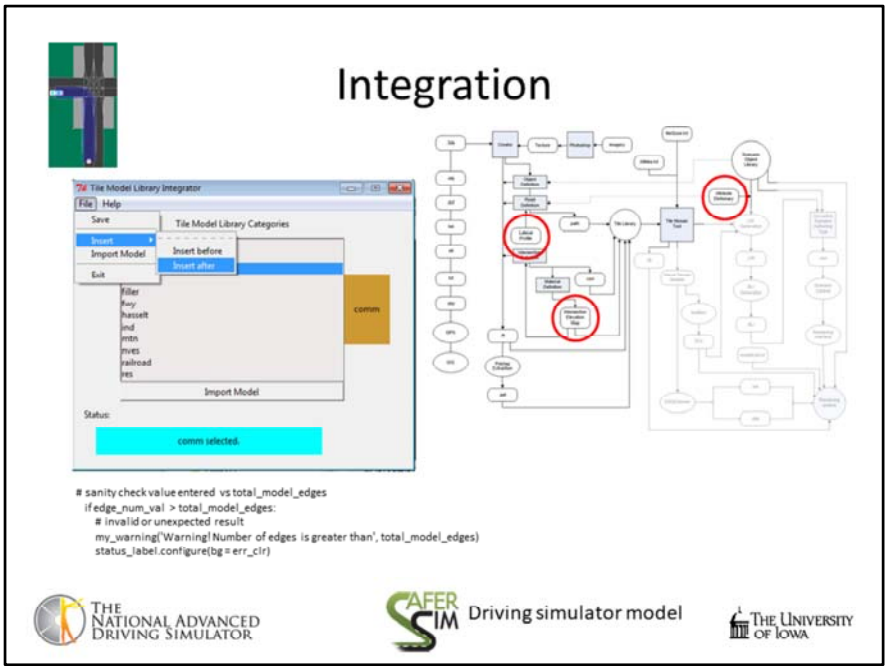
The second tool view shows the intersection layout is complete enough to look like it can support traffic.

In this version we added textured geometry in the visual model. This is a screenshot showing how texture has been applied to the model, and also where some geometry was lost in the process (shown in the orange area).



The converter generates a large number of files, most of which need to be placed into their respective folders or files within the model library framework. This slide shows the processing pipeline from data resource on the left, through various processes until getting to the simulator in the lower right corner. I've drawn an outline around the area not directly related to the model integration process.

The remaining left half shows how much of the simulator pipeline is being managed by the converter, starting on the far left with data input types, then following conventional process steps to get files integrated for use with the tile layout application.

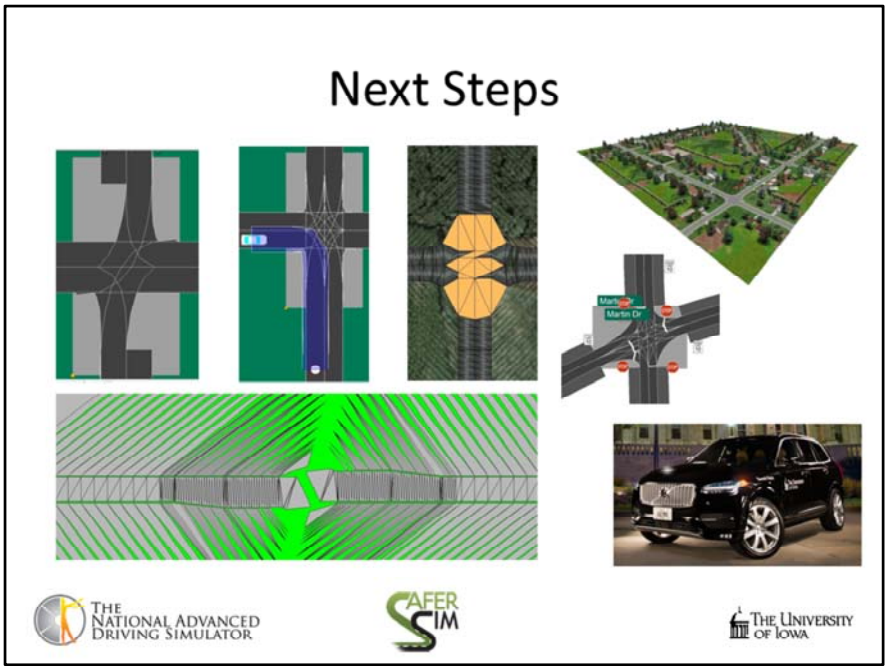


Fortunately there is another tool to help integrate converter output. This tool is a python script for managing the tile layout library. Although the converter is currently a command line tool, it could be added to this to consolidate the number of steps required.

The highlighted files show where the user still has to manually merge data.

This GUI tool allows the user to select a file to import into a model category, and then it performs some file handling and data processing steps. One of these steps is to define the size of the tile model, which is crucial for some of the processing downstream on the way to the simulator. Each tile model contains some meta-data in the file header that is normally inserted with a 3D editor.

This tool guides the user to create the proper data, manages syntax and provides color coded feedback.



Several weaknesses have been introduced:
Intersection processing needs to become more robust. In the screenshot you can see the roads are symmetric, but there is a vertical bias to the intersection. A more reasonable definition of the junction area would be more square than horizontal or vertical.

The region of connectivity needs to be modelled better. Currently the connectors extend beyond road edges, perhaps due to the region of intersection being too small. This causes problems for simulator traffic to navigate the intersection without a problem.

The algorithm for processing intersection geometry completely breaks down, and leaves a hole. In the example here the hole has been filled in manually and left un-textured.

The geometry processor needs to be more robust to produce a complete surface skin. The screenshot here shows a significant number of triangles do not make it into the output.

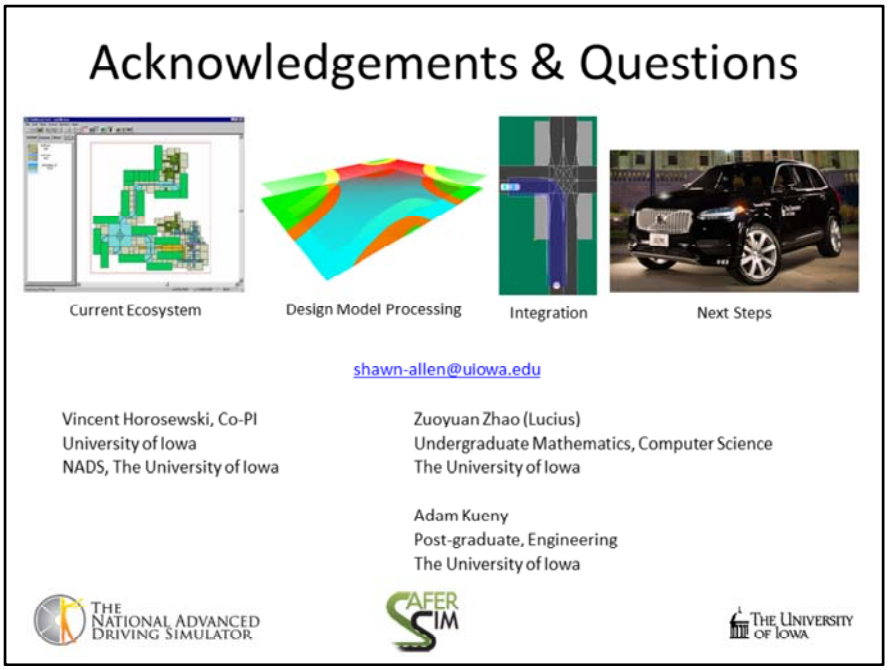
There are some issues in the TIN processor that needs to be sorted out in order to support design model objects. In this case objects could be anything from road markings to signs, barriers or medians.

The converter is very nearly complete in terms of compatibility with the Tile Model Library in terms of how various data files are produced. However, a few things remain to be added.
In order for tile models to connect to one another, some conventions are used for model size and position. Right now the converter does not accommodate these conventions, so the output is effectively a copy of the design model input.

The converter needs to be able to extend the design model in size to satisfy library conventions, and then translate the converted geometry to a local origin of 0,0,0. This will ensure the model is compatible with the rest of the tile library.

Finally, the notion of a tile library is built around the notion that models are built once and re-used many times. There's really no convenient work process to deal with objects in terms of a timeline or repetitive development cycle. Currently the integration methods and tools are one-way INTO the system, and this needs to become more flexible.

Probably the simplest way to deal with this is to move the NADS model library from an ad-hoc data file structure to a true relational database to manage transactions at the model level.



I'd like to acknowledge the sponsor of this development project, the SaferSim program and the US DOT.
I would also like to thank my co-PI, Vince and our students Adam and Lucius for their contributions with design models and programming support, respectively.