AUTOMATIC CORRELATED TERRAIN DATABASE GENERATION AND MANAGEMENT FOR GROUND VEHICLE SIMULATORS

Yiannis Papelis, Shawn Allen, Ben Wehrle

Simulator Technology Branch
National Advanced Driving Simulator & Simulation Center
The University of Iowa

Abstract
Modeling and interrogating the elevation of the terrain is necessary in both flight and driving simulation applications. In flight simulation, with the exception of ground based operations, the requirements for resolution and interrogation rate can often be met by the image generator or other simplified databases containing airports and the surrounding area. In ground vehicle simulators, however, the requirements for interrogation rate, performance, resolution, and geographical coverage are much more stringent and separate databases are often used. Work described in this paper extends prior developments on correlated terrain databases specifically focused on driving simulation applications by adding key features on usability, correlated data generation, singularities, and performance. Key features of the earlier system included variable resolution storage, real-time execution time independent of the size of the database, and the ability to model vertically stacked terrain with minimal runtime overhead. Improvements of the new system include the automation of the database generation process given the source code of the visual databases, a new interrogation technique that eliminates distracting artifacts that appear across areas that have been sampled at different resolutions, and the use of graphical interactive tools for verifying the correlation of the final terrain database against the original visual model. The paper includes examples that demonstrate the process, along with information on the performance of the code under various conditions and computing systems.

Introduction and Prior Work
In ground vehicle simulators, where the interaction of the vehicle with the environment provides the majority of the feedback cues, a representation of the underlying terrain is critical for proper fidelity and realism.

Terrain databases used for ground vehicle simulator applications have stringent requirements that eliminate the use of the Image Generator (IG) as a viable alternative for obtaining height above terrain (HAT) information. For example, a vehicle model executing at 240 Hz and requiring one interrogation at each wheel is equivalent to a 960 Hz interrogation rate for the terrain database. Given that at 240 Hz, the dynamic model has no more than 4.16 milli seconds of real-time available to perform all computations, the terrain interrogation must be very efficient, preferably taking no more than 1% of the total available time, or approximately 40 micro seconds. More complicated vehicle models often require even higher update rates [1] (i.e., 500 Hz or more) and multiple axis vehicles such as trucks require even more interrogations due to the larger number of contact points. Further increasing the requirements are sophisticated tire models that require the terrain elevation along a two dimensional patch as opposed to a single point [2], which in turn dramatically increases the number of interrogations that need to be performed per iteration of the dynamic model. In addition to the stringent performance requirements, ground vehicle simulator systems also necessitate the use of very high resolution data. Depending on the complexity of the tire model, resolutions as fine-grain as five or ten centimeters are necessary. Whereas visual databases often utilize textures to provide high resolution features, even though the underlying polygons are often larger than 125 meters, the actual terrain database must be able to represent high resolution features with distinct elevation and/or surface property feature sets. It is clear that the cumulative requirements of high fidelity ground vehicle simulation systems as described above necessitate terrain databases that are specifically designed to address these requirements.

Prior work on this area has focused on using regular grids for storing elevation and surface features, but with different capacities for dealing with large databases, and different interrogation algorithms [3, 4] for different applications. The approach described in [3]
revolved around the basic notion of using multi-resolution storage that allows optimal use of space for representing large databases with different resolution requirements. The basic concepts involved in the representation of the terrain are reviewed here but further details can be found in [3].

Terrain elevation, in its basic form, is provided by a uniform grid of elevation posts. Each post, often called a dataset, contains the elevation of the terrain at a given point along with information about the surface. The actual information varies with the application but typical items in simple applications include friction coefficients and various constants used to compute the forces upon the tire. In more complex applications, items stored in datasets include information about water bodies (i.e., depth, flow) along with soil properties at various depths. From the standpoint of the terrain database, each dataset represents a unit that is managed independent of its size or contents. Datasets are placed along a uniform grid within a rectangular area. The spacing of the grid can be different along the x-axis and the y-axis, but is uniform so that locating the four datasets that surround a query point is an operation that can be performed very efficiently and is independent of the size of the grid. The distance between datasets is the resolution. Note that in this context, the term high resolution refers to a smaller grid spacing providing more datasets per unit area. A group of datasets within a rectangular grid is called a datazone. Variable resolution storage is achieved by utilizing multiple datazones, each with a different resolution. Figure 1 illustrates a simple example of three datazones occupying different areas on the x-y plane. Note that datazones are rectangles whose sides are aligned with the x and y axis. Each intersection of lines within a datazone represents a location on the x-y axis that would be occupied by a dataset. So a datazone with n rows and m columns contains m * n datasets, and the dataset's x and y coordinates are implicit given its row and column position within a datazone.

Datazones can overlap. Overlapping datazones can represent overlapping terrain such as bridges or tunnels. When datazones overlap, it is not necessary for all of their datasets to provide terrain that is vertically spaced. In fact, it is possible to have overlapping datazones that contain data representing the same elevations. This often occurs when a datazone is larger than the elevated feature it represents, for example a bridge. The datasets representing the elevated feature will contain the new elevation, but the datasets that fall to the side of the bridge should in fact contain the same elevations as the datasets of the datazone representing the underpass.

**Interrogation Algorithm**

The input to the interrogation algorithm is a location on the x-y plane represented by a point P(x,y) along with an estimate of the z elevation. Thus the input to the query is a triple (x, y, z). There are three steps involved in determining the elevation of the terrain as follows:

1) Find the set of datazones that overlap point P.
2) For each datazone, do the following computations, storing the results internally:
   2.1) Compute the coordinates of point P with respect to the datazone origin, store in P'.
   2.2) Determine the row and column of the four datasets that surround P'.
   2.3) Use bilinear interpolation to compute the elevation at P'.
   2.4) Based on proximity to the four surrounding elevation points, pick the best dataset for representing the surface properties.
3) Given the initial estimate of z, pick the best candidate among the ones found during step (2).
4) Use bilinear interpolation to compute the normal vector at point P', given the best candidate.

The complexity of step (1) depends on the data structures used to aid the searching of datazones given the query point. In the worse case scenario, a linear search with O(N) complexity can be used (where N is the number of datazones in the system). Two dimensional search data structures can easily be used to reduce that complexity. Quad trees can reduce the complexity to O(log(N)), or as shown in [3], two dimensional hash tables can reduced the complexity to the point where it is independent of the number of datazones in the system, or O(1). The computational
complexity of Step (2) is independent of the total size of the database, the resolution, or the dataset size. For example, step 2.1 simply involves a few arithmetic operations (primarily subtractions), step 2.2 involves a few divisions and truncation operations, step 2.3 involves the bilinear formulas and finally step 2.4 involves few comparisons and data movements. The only O(N) complexity is due to multiple overlapping datazones. Generally, even though O(N) is not considered acceptable for real-time applications, the upper bound on N is the maximum number of overlapping surface in a datazone. In general, this upper bound is very low, usually no more than two or three, in off-road applications. Finally, steps (3) and (4) are also O(1) complexity involving few arithmetic operations that are independent of the size of the database.

In summary, provided that an efficient two dimensional search data structure is used for step (1) and a reasonably small bound can be placed on the maximum number of overlapping datazones, the interrogation algorithm has fixed order computational complexity thus is independent of the resolution of the datazones or the overall size of the database.

Runtime Management of Data

Given arbitrarily high resolution, terrain databases can often become too large to fit in the main memory of the computer running the dynamic model. In such cases, a runtime component is used to page in data necessary for terrain interrogation before they are needed by the dynamic models. The runtime component is using a prediction algorithm to estimate the datazones that will be needed by the dynamic model and load them in memory before the interrogation code actually needs them. A detailed analysis of the bandwidth requirement for various terrain types and densities was done at [3], but for clarity a small summary is repeated here. The data throughput required between the disk I/O subsystem and main memory to transfer data fast enough to keep up with a sweeping line whose length is equal to W, and is traveling at a velocity of v is equal to

\[ n \cdot \frac{W \cdot v}{\text{res}^2} \cdot \text{DSS} \]

The DSS symbol represents the size of the dataset, in bytes, res represents the resolution of the terrain, and n represents the average number of overlapping datazones over the sweep area. Note that the basic bandwidth increases in the presence of overlapping terrain. In applications where it is necessary to write back the data, such as when working with dynamic terrain, the bandwidth will double. An additional consideration is the granularity of data as it is moved in and out of memory. If whole datazones are paged, there is significant potential for external fragmentation due to the varying size of datazones. If smaller transfer units, such as datasets are used, then the bandwidth will suffer and hard disks operate best when transfers take place in disk block units. Finally, using a transfer unit that does not correspond to a basic object in the terrain database, for example a 4096 byte block, then there is significant computation involved in determining the relation between the needed geographical coverage, and the actual data that has to be moved within the block.

Enhancements

Despite its usefulness as an efficient terrain interrogation system, there are several limitations that became apparent following extensive use in various ground vehicle simulators. These limitations include the inability of the model to efficiently represent vertical edges, discontinuities across datazones of different resolution, and the challenging process by which such databases have to be constructed, especially when a corresponding visual database already exists. Work presented in this paper addresses these limitations.

Discontinuities due to elevation

One of the disadvantages of the elevation grid representation is that it cannot effectively represent vertical terrain profiles as often encountered in sidewalk edges or river beds. Generally, if the edge of such an object is aligned to the x or y axis, adjacent datazones whose datasets have different elevations can be used to represent the discontinuity.

![Figure 2 - Edge Object Profile.](image)

When the features are not aligned to the x or y axis, vertical edges have to be represented by changes in the elevation on adjacent datasets. The maximum angle edge that can be represented by adjacent datasets is limited by the resolution of the datasets, with the extreme limit being a vertical edge which cannot be represented by elevation posts since it would require a
zero resolution. Unfortunately, such edges are very often encountered both in off-road and on-road applications. When off road, rock formations or river beds often require modeling vertical edges. Even though our focus is for off-road applications, it is interesting to note that even when on-road, curbs and sidewalks are very common and would pose the same limitations on any terrain model that depends on uniformly placed elevation posts.

To address this limitation an added layer of special effect edge objects are used to represent such terrain. An edge object consists of a series of connected line segments that all lie on the x-y plane, each segment representing the edge of a discontinuity of the terrain elevation. The discontinuity is represented by several parameters: dl, d2, d3, h and a base height. Figure 2 illustrates how these parameters model the profile of an edge. In that figure, a line segment would be perpendicular to the page. The base height establishes the elevation of the bottom area of the edge object. This allows the modeling of uneven terrain with arbitrary drops and independent of the resolution of the underlying elevation grid. Note that selecting a value of 0 for the parameter d2 yields a vertical edge.

The interrogation algorithm described earlier has to be slightly modified to accommodate edge objects. When the position being queried for is inside the d1 segment of the edge object (point P2), the terrain query interpolates between the height of the terrain grid and the base height of the object. At point P1, the query returns the height of the terrain grid, and at point P3, the query returns the base height of the edge object. When the position being queried for is inside the d2 segment of the edge object (point P4), the terrain query interpolates between the base height and the top height of the edge object. Finally, when the position being queried for is inside the d3 segment of the edge object (point P6), the terrain query interpolates between the top height of the edge object and the height of the terrain grid. At point P5, the query returns the top height of the terrain grid, and at point P7, the query returns the height of the terrain grid.

When the line segment defining the edge object does not lie in the x-y plane the terrain query determines the elevation in two steps. It first assumes that the object does lie in the x-y plane and performs the same calculations as before. It then adds to this calculation a linear interpolation between the lower and upper points of the line segment to account for the segment.

**Datazone Boundary Discontinuities**

When traveling across datazones with different resolutions, the sampling of the terrain may yield different results on the edge, depending on which datazone was used. This problem is amplified in areas where the resolution is not high enough to capture all the effects of the underlying terrain. Whereas small errors due to undersampling are acceptable, the discontinuities due to different datazone resolutions are not acceptable as they often cause jerks to the steering wheel or force the generation of sharp accelerations by the simulator's motion system. To resolve such errors the transition from the boundary of the current datazone to that of the next must be made more gradual. When a query is made near the edge of a datazone that is adjacent to a datazone with a higher resolution, a special smoothing interpolation is used as follows. Two new elevation posts are dynamically created at a distance consistent with the grid of the higher resolution datazone, but within the datazone with the lower resolution. The elevation is then determined using bilinear interpolation using the new "virtual" elevation posts of the current datazone. The actual interrogation results are computed using the two virtual datasets along with the datasets from the adjacent datazones.

This process is illustrated Figure 3. Two adjacent datazones have different resolutions. The one on the right has a higher resolution than the one on the left. Let us refer to the resolution of the left datazone as Rl and the resolution of the right datazone as Rr. An interrogation is made at point P, which falls on the datazone on the left, within the quad defined by points p1, p2, p3, and p4 or any other adjacent dataset on the left datazone. That point is located less than Rr units away from the edge of the right datazone, in effect
triggering the generation of two virtual elevation posts. These new virtual posts, \(vl\) and \(v2\) are computed by interpolation between \(p1, p2, p3,\) and \(p4\) but on a location that is consistent with the spacing of the high resolution datazone on the right. Once points \(vl\) and \(v2\) are calculated, the interrogation is performed using the quad consisting of points \(vl, v2, k1\) and \(k2\). Note that by the nature of the bilinear equation constraints, queries along a shared edge of two adjacent quads yield similar results no matter which quad was used. Use of the virtual elevation posts ensures that the transition between the two datazones becomes continuous even when there are large differences between the sampling rate of adjacent datazones. One disadvantage of this technique is that additional data structures have to be used to ensure that the computational complexity of the interrogation algorithm does not suffer due to the need to determine if adjacent datazones exist. Currently, a prototype system is under development and preliminary results indicate that use of simple adjacency lists provide a very efficient method for finding adjacent datazones. In this context, an adjacency list is a list associated with the edge of each datazone containing any adjacent datazones with higher resolution. The two additional bi-linear interpolation calculations are additive in nature, in effect increasing the computation time by a fixed amount, but they are still independent of the total size of the database.

**Automatic Generation of Terrain Databases**

One of the key challenges in the effective utilization of a terrain database system is the ease by which a terrain database can be constructed, and the final degree of correlation between the terrain data and the original visual database used as a source. The concept behind the automated terrain database extraction is simply to interrogate the visual database polygons and extract the elevation grid which is used at runtime. The actual process involves quite a few translation details along with various algorithms for determining the datazone layout and elevation post contents. These issues are further described below.

A block diagram of the terrain database generation process is shown in Figure 4. The process begins by extracting a list of polygons from a visual database. Only polygons representing the terrain skin are considered at this point. The actual method by which terrain polygons are discriminated from other polygons that represent features varies depending on the format of the source file and various implementation issues, but most formats provide for simple methods by which the actual polygons and their attribute information (i.e., texture names, etc.) can be dumped into an ASCII file.

![Figure 4 - Terrain Database Generation Process.](image)

The current prototype incorporates parser implementations for the Evans & Sutherland General Database Format (GDF) and the MultiGen OpenFlight format. Additional formats, such as OpenInventor, or Sedris can easily be accommodated if necessary. The visual database polygons are triangulated (if necessary) and stored in a TRI file. Each line in this file contains the x, y, and z coordinates of each of its vertices, and any type specific information. Type information includes the surface properties associated with a triangle, or additional meta-data that may affect its interpretation. For example, lake beds or polygons representing water surface are identified as such allowing proper interpretation and storage in the dataset.

The user must also supply a datazone layout, or DZL file. A DZL file is a list of boundary-aligned rectangles along with a required resolution that describes the proposed datazones for the final datazone. Each line of the file contains the x and y coordinate for the lower left and upper right corners of the datazone, and the number of divisions along both the x and y axis. At the prototype system, this file is generated by hand, but work is underway for automating this phase of development with automated tools that use heuristics and optimization algorithms for automatically determining an optimal, based on user specified criteria, datazone layout.

The tagged triangles and DZL file are fed to the scanner program which is responsible for generating the datazones and associated datasets. The scanner operates...
as follows. The polygons are first loaded into memory from the TRI tile, and their projections onto the x-y plane are stored in a quadtree lookup table. The terrain is then broken up into datazones as specified by the DZL file, and each datazone is broken up into a grid of datasets, or elevation posts. In order to construct the final datazone the elevation, and the type specific information must be determined for each elevation post within the datazone. For each elevation post the quadtree returns all polygons and objects that contain the coordinates of the elevation post within its boundaries.

One of the challenges facing the scanner is the proper generation of overlapping datazones when the multiple polygons, each representing a different overlapping surface, exist within the boundary of a datazone. To detect and label overlapping triangles the sampler keeps a reference of the triangle used to determine the elevation of the previous elevation post. The sampler also maintains a list of any triangles that have been tagged as overlapping within the datazone. The algorithm of selecting which triangle to use to determine the elevation of the post proceeds as follows:

1) The quadtree returns all triangles that contain the elevation post
2) Any triangles in the quadtree list that are also in the overlapping list are removed from the quadtree list.
3) If any of the remaining triangles match the triangle from the previous elevation post
   A) Select that triangle
   else
   B) Select the triangle with the lowest elevation
4) Use the selected triangle to determine the elevation and characteristics of the post
5) Remove the selected triangle from the quadtree list
6) Any triangles that remain on the quadtree list represent overlapping terrain and are added to the overlapping list. As an exception to this step, if the elevation of a triangle in the modified quadtree list is less than some specified tolerance from the elevation of the selected triangle, then they are said to share the elevation post and are not considered overlapping. This allows multiple triangles to share a common vertex or line segment.

If the overlapping list contains at least one entry once the characteristics of all of the elevation posts have been determined then the datazone contains overlapping terrain. At this point the sampler creates a second quadtree that contains only those triangles that were on the overlapping terrain list. Once the new quadtree has been created the overlapping terrain list is cleared so new overlapping triangles may be detected. A new datazone is also created with the same bounding box and elevation post densities as the previous one. The new datazone then queries the new quadtree to determine the characteristics of its elevation posts. The same steps as before are used to select triangles, and the process continues until the overlapping terrain list is empty at the end of a datazone. At this point all levels of overlapping terrain have been modeled and the sampler continues to the next datazone in the DZL file.

Figure 5 illustrates an example. In this case, neither triangle A nor triangle B are on the overlapping terrain list, but triangle A matches the previous triangle so it is used to determine the elevation of the post.

Figure 6 illustrates the operation of the algorithm during the generation of another pair of elevation posts. In this case, triangle B is on the overlapping terrain list and is removed from the quadtree list. Triangle A matches the previous triangle so it is used to determine the elevation of the post.

The final step in the process of generating the terrain database is the compilation step. The compilation step is responsible for pre-computing all the internal data structures used during the interrogation algorithm for searching applicable datazones. Also, the adjacency lists which are used to determine if virtual elevation posts need to be computed are also created during this step. Finally, datazones that exceed the maximum size required by the lookahead algorithm are broken into multiple smaller datazones in such a way as to minimize internal fragmentation. The input to the compiler is simply a list of datazones and associated
datasets, and the output from the compiler is an integrated terrain database file that can be used as input to the interrogation code, or can be automatically managed by the run time lookahead loader for inclusion in a real-time simulation system.

The system described in this paper is currently under construction. Several prototype implementations have been coded to experiment with execution speed using various methods for the two dimensional lookup functions, and for validating the expected size independent execution speed. At this point, the final system is nearing completion, with several of the components already coded and tested. A parser for the E&G GDF format has been coded and tested and an OpenFlight parser is underway.

One of the challenges encountered during the implementation of the GDF parser is parsing databases that are programmatically specified. Specifically, databases that use basis sets provide a list of templates that during compilation of the visual database are placed at different places in the database while at the same time adjusting their vertices to snap onto existing terrain. This feature minimizes modeling time since it allows the re-use of cultural features and vegetation templates across large areas without the need to manually apply each and every feature. However, use of basis sets eliminates the ability of a tool to extract the terrain skin from the source code of a database. Instead, the parser tool will have to operate on the final binary file as prepared for the image generator, which already contains the instances of the various templates as they will appear on the IG. Some additional corrections have to be made to adjust the triangles to accommodate their modified position due to the snapping onto the surrounding terrain, but with these modifications, the tool performs as expected.

The scanner program is also completed and currently undergoing testing. To evaluate the operation of the algorithm that discriminates overlapping terrain and produces multiple datazones to represent vertically stacked terrain, several reference databases were created. One of the most challenging ones is the representation of a multi-level parking ramp as shown in Figure 6. In addition to providing a challenging test for the scanner, this reference database allows us to determine the sensitivity of the execution time to multiple overlapping surfaces.

Finally, the compiler and interrogation code have been coded and tested. The runtime lookahead manager is currently under development.

The software for the off-line tools is written in C++ and has been compiled and tested in several Unix and PC platforms including IRIX, SunOS, Linux and WinNT. Similarly, the interrogation code is written in C and is highly portable. To this date, it has been compiled in all the systems that the off-line tools are operational, but in addition, it has been compiled in the VxWorks operating system as part of simulators whose software runs on embedded system using that operating system.

The most critical performance number is the execution time of the interrogation code, which along with the lookahead loader is the only component that runs in real time. The assumption that the execution code is independent of the size of the database has been experimentally validated. A set of experiments were run to measure the execution speed of the interrogation code. The total database size is approximately 50 mega-bytes, and the datazone search algorithm (described in step (1) of the algorithm) is using a data structure that searches across 10 datazones before finding the applicable ones. This is a very conservative estimate since in practice, the search is often limited to no more than three or four datazones. Several configurations were tested. In the first configuration,
all query points were on areas of the database that had no overlapping terrain. In the second configuration, the interrogation point was on a location containing two overlapping surfaces. The difference in execution time between these two cases represents the computational load of the bilinear interpolation calculations. Another factor entering the experiment is compiler optimizations. Generally, depending on heavy compiler optimizations makes the software less portable, however, due to the nature of the bilinear interpolation, the potential for severe pre-calculations, and the ability of better using the pipelines often encountered in modern processors, the experiments did include measurements of compiler optimized code.

The following table summarizes the execution times that were obtained per interrogation. Multiple tests were run, and the average is reported. The column labeled NOOV, NOOPT contains execution times for no overlapping surfaces and with no compiler optimizations. The column labeled NOOV, OPT contains execution times for no overlapping surfaces but with compiler optimizations turned on. The column labeled OVLP, NOOPT contains execution times for two overlapping surfaces with compiler optimizations turned off, and finally, the column labeled OVLP OPT contains execution times for two overlapping surfaces with compiler optimizations turned on. No special effect objects were used in this experiment. All units are micro seconds (1E-6 seconds).

<table>
<thead>
<tr>
<th></th>
<th>NOOV NOOPT</th>
<th>NOOV OPT</th>
<th>OVLP NOOPT</th>
<th>OVLP OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mips R44k, 250M</td>
<td>12</td>
<td>6</td>
<td>19</td>
<td>9</td>
</tr>
<tr>
<td>Mips R5k, 195M</td>
<td>11</td>
<td>5</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td>Mips R10k, 195M</td>
<td>8</td>
<td>4</td>
<td>11</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 1 – Interrogation Performance.

Conclusion and Future Directions.
This paper describes an integrated process for generating, managing and interrogating variable resolution terrain databases used primarily in ground vehicle simulation applications. Various algorithms and techniques for generating data that can be for the real-time interrogation of the terrain at very high speeds were described along with the rational for the various chosen design decisions.

Several areas can benefit from improvements in the existing processes. Generating the DZL files is at this point a manual processes, sharply contrasting the automation found in the remaining of the system. Furthermore, even when given enough time, it is hard to manually estimate an optimal datazone layout. Clearly, even when given enough time, it is hard to manually estimate an optimal datazone layout. Clearly, a much needed improvement is the development of a tool that based on some optimization criteria can automatically generate an optimal datazone layout. Some of the criteria may include minimizing the overall storage requirements, minimizing the sampling error, or minimizing the number of datazones.

Another area that can benefit from additional work is the determination of the surface properties, once a visual database polygon has been identified as the underlying terrain skin. Currently, the surface properties are calculated based on a lookup table that given the texture and/or color of the triangle, provides the necessary surface properties. However, a popular visual database modeling technique is using global textures for adding details without using too many polygons. Global texture is a term that refers to texture that contains features drawn within it. For example, a single texture may contain a lake, roads, and forests, however, given the current technique, all queries within a polygon using that texture would be associated with the same surface properties. An alternative would be to determine the actual texel contained on the location of the elevation post and the use the texel value to determine the surface properties associated with that elevation post. Finding efficient ways and developing manageable logistics to map texel values to surface properties is an open issue at this point.

References