Producing Terrain Databases for Computer Generated Forces using SEDRIS

Victor J. Skowronski
Thomas Stanzione
TASC, Inc.
55 Walkers Brook Drive
Reading, MA 01867-3297
Tel: 781-942-2000, ext 2728
Fax: 781-942-0636
Email: vjskowronski@tasc.com

Keywords:
Terrain Representation, Modeling and Simulation

ABSTRACT: The Synthetic Environment Data Representation Interchange Specification (SEDRIS) provides a common interface to a variety of sources of synthetic environment data. This paper describes how SEDRIS has been used in the creation of a compiler to convert SEDRIS data to the format of the Compact Terrain Database (CTDB) used by the ModSAF Computer Generated Forces (CGF) program.

SEDRIS uses a hierarchical data structure that allows the data producer a large amount of flexibility in organizing his data. If he knows the data producer's organization scheme, a data consumer can construct very efficient code for reading a transmittal. However, this code may not work with SEDRIS transmittals that have a different, but equally valid, organization. A design goal of the compiler described in this paper was to be able to accommodate any data organization that might be valid in SEDRIS.

A second design goal of the compiler was to make maximum use of existing code. In particular, the previous, specialized compiler had been divided into two sections, a front end that read data from the source database, and a back end that created the interdependent structures of the CTDB. The back end is independent of the source data's organization, so it was especially suitable for reuse.

The resulting compiler attempts to extract objects from the SEDRIS data that correspond to atomic objects recognizable by the compiler back end. Additional processing is needed to present these objects to the back end in an appropriate order and to extract them from the SEDRIS transmittal in an efficient manner.

The compiler has been tested with SEDRIS transmittals based on Lockheed-Martin's S1000 database format.

1. Introduction

The Synthetic Environment Data Representation Interchange Specification (SEDRIS) provides a common interface to a variety of sources of synthetic environment data. This paper describes how SEDRIS has been used in the creation of a compiler to convert SEDRIS data to the format of the Compact Terrain Database (CTDB) used by the ModSAF Computer Generated Forces (CGF) program.

The next two sections describe the relevant features of SEDRIS and CTDB respectively. The strategy for converting SEDRIS data to the CTDB format is then discussed. The final section describes the results of compiling two databases from the Lockheed-Martin S1000 format using the SEDRIS compiler.

2. SEDRIS Overview

SEDRIS defines a non-proprietary data representation. This representation is accessed through an Application Program Interface that allows the user to extract objects of interest. The next two sections discuss the organization of data in this representation and the application program interface. The final section describes how the data representation and interface support the
CGF terrain compiler in reading data from many formats.

2.1  SEDRIS Data Organization

The highest level SEDRIS data object is the transmittal. A SEDRIS transmittal is a collection of objects. These objects are organized hierarchically. Objects that contain other objects are called Aggregate Objects. The objects contained are called Component Objects. Component objects may also be aggregate objects.

At the top of every SEDRIS transmittal is the synthetic environment. The synthetic environment aggregates the rest of the objects in the transmittal. Directly beneath the synthetic environment lie various libraries and the Base object. The Base object in turn contains two hierarchies, a geometry and a feature hierarchy. The feature hierarchy contains objects that represent a higher level of abstraction than those in the geometry hierarchy.

The geometry and feature hierarchies are both types of a more general type of SEDRIS object called a Union. Unions organize data and are discussed in the following section.

2.1.1  Unions and Aggregations

Most SEDRIS objects aggregate other objects. However, a few objects have been specifically created to aggregate objects according to some particular parameter. These aggregating objects are called Unions. Unions may group objects according to location (Spatially Indexed, Quadtree, and Octtree Unions), common feature (Classification Related Unions), time dependence, or state dependence. Each component of a union has a Link Object associated with it. The Link Object specifies the value of the grouping parameter of the union that applies to this component. For example, the Link Objects for a Spatially Indexed Union would describe the spatial extents of the component object with which the Link Object was associated.

2.1.2  Attributes and Data Storage

Objects exist to store data. A SEDRIS object stores its data in its attributes. The attributes for each object are organized in a specific data structure unique to that object. The structure is defined in the SEDRIS Application Program Interface so that individual items of data can be read directly from the attributes list of an object.

However, not all the data needed to define an object is stored in that object’s attributes. Much of it is stored in the attributes of component objects or link attribute objects. Some data may also be stored at the level of a component’s aggregates or through associations with other objects.

2.1.3  Associations

Associations are provided for connections between SEDRIS objects that cannot be expressed as an aggregation of components. One example of this is a model. A Model is a collection of objects that may be instanced in several locations within the synthetic environment. A Model Instance object specifies the location. The model instances are aggregated within the geometry or feature hierarchies, while the Model is part of the Model Library. The connection of the model instance to the model is via an association.

2.2  SEDRIS Application Program Interface

SEDRIS objects are accessed via the SEDRIS Application Program Interface (API). The API contains routines and data structures that automate the traversal of the SEDRIS data hierarchy. The data structure that controls a traversal is the Iterator, which is discussed next.

2.2.1  Search Iterators

The SEDRIS iterator contains the user’s instructions concerning the way a SEDRIS hierarchy is to be traversed. The data includes the starting object, the links to be traversed (component, aggregate, or association), objects to be extracted, and the geographic bounds of the search. The user can also specify whether information is to be inherited from aggregate objects. Inheritance is important because it makes information about an object that was stored at a higher, aggregated level available along with the object. Some of the iterator instructions are supplied directly as arguments when the iterator is initialized. Other instructions are encoded in search rules that are then passed to the initialization routines.

2.2.2  Search Rules

Search rules provides a flexible means of specifying the objects to be returned. The rules can be divided into two types, object tests and booleans. The boolean rules combine the results of the object tests as required by the user. Object tests specify the characteristics of the objects that are to be returned. SEDRIS object type can be specified as well as objects with certain data values.
2.2.3 Geographic Search Bounds

Since most applications prefer to process objects by region, geographic search bounds are provided to limit a search to objects within a particular area. SEDRIS allows the user to define a three-dimensional rectangular box as a search bound. Depending on the option chosen, objects contained within the box, partially contained within the box, or possibly contained within the box will be returned, providing the object meets the other search criteria.

2.3 SEDRIS Support for CGF Data Requirements

The Compact Terrain Database (CTDB) compiler requires that objects to be compiled be self-contained (no cross-referencing), and be readily identifiable as to what they represent (terrain polygons, buildings, trees, etc.). This allows the compiler to work on objects one at a time, and to process each object in the appropriate manner.

SEDRIS supports these CGF data requirements by providing a set of objects that correspond to objects used by the CTDB. Each object acts as an aggregator of the data needed by the CTDB to create that object in its database. The needed data is stored in the SEDRIS object’s components, associations, or link objects.

SEDRIS also supports the CGF data requirement for identifying object type through the Classification Data Object. The Classification Data Object is either a component or a Link Object of the SEDRIS object being classified. It specifies what an object represents in the environment. For example, a terrain polygon has a specific classification data value and a canopy polygon has a different one. These values can be tested to determine how the object should be compiled. The Classification Object values are codes from the Features and Attributes Coding Catalog (FACC) and have been chosen to correspond to a particular type of object in the environment.

The classification data object provides the means to extract the correct objects from SEDRIS regardless of the source data organization. Search rules allow the user to extract objects with particular FACC codes and ignore others. This allows the user to search for objects based on their classification data with some assurance that the proper objects will be identified.

3. CTDB Overview

This section provides a brief description of the principle types of data stored in the Compact Terrain Database (CTDB). (For a more detailed description, see References [1] and [3].) The types of data described are terrain elevation and soil grids, physical features, abstract features, and models.

3.1 Elevation and Soil Grids

Elevation and mobility data is stored in two arrays. The elevation array stores the elevation of the terrain at each point of a square grid. The mobility or soil type array stores soil data for two right-triangular polygons to the southeast of each grid point of the elevation array. The grid spacing and origin can be adjusted so that the grid aligns with any grid that might exist in the source data.

3.2 Physical Features

Physical features are stored according to their location. The entire database is divided into squares of 500 meters on a side called patches. Each patch stores information about the features located within it. The information includes the exact location or locations of the feature, its mobility code, and other information specific to the type of feature. Physical features that overlap patch boundaries are divided and the appropriate section is stored in each patch data structure.

3.3 Abstract Features

Abstract features are stored in a quadtree structure. This allows for features larger than a patch. Abstract features are also more complex than physical features. For example, soil defragmentation areas are permitted to have “holes”, areas within the perimeter of the abstract feature where the properties of that feature do not apply. In CTDB, this is implemented by overlaying abstract features on top of the base feature, and assigning priority numbers to the features. In SEDRIS, the areal feature has multiple perimeters, with all but the first perimeter defining the holes in the feature.

3.4 Models

A model is a collection of data that is common to more than one feature. In order to conserve memory, model data is stored in a model library. When the model is “instanced”, inserted into the terrain, the location is recorded along with a reference to the model in the library. CTDB models typically consist of a set of
roofline vertices that describe the height and the perimeter of the model structure. These models are essentially obstacles. However, a special type of model, a Multi-Elevation Structure (MES), has been developed. The MES has interior enclosures that can be traversed (see Reference [2]).

CTDB uses models for both buildings and bridges. Buildings may be modeled as either a conventional volume model or as a MES. Bridges are always modeled as an MES. Otherwise they cannot be crossed. The back end of the CTDB compiler contains routines to generate the appropriate data structures for simple MES buildings and bridges. This reduces the need to decode complex data structures in the source data.

3.5 Compiler Organization

The CTDB compiler is organized in two sections, a “front end” and a “back end”. The front end reads data from the source database and puts it in an order that can be processed efficiently. The front end also provides a standard format for the data. The back end receives the ordered and formatted data from the front end and creates a CTDB database.

In the design of the new compiler, the function of the back end has remained unchanged. However, the front end was redesigned to accommodate the new SEDRIS API (see Figure 1). The next section describes compiler operation in more detail.

4. Translating SEDRIS to CTDB

The philosophy that was employed to translate SEDRIS data to CTDB data was to identify SEDRIS objects that corresponded to data objects in the CTDB. Such objects are elevation readings (for grid posts), polygons (for terrain), and primitive features (for both physical and abstract features). Once such an object was identified, the data needed by the CTDB had to be extracted, converted to a suitable format, and then passed to the compiler back end.

The following sections describe this process in more detail. The first step is to navigate through the SEDRIS hierarchy.

Figure 1. Compiler Processing
1.1

4.1 Traversing the SEDRIS Hierarchy

The compiler performs two traversals of the SEDRIS hierarchy. The first traversal takes place as part of the metadata processing (see Figure 1). It collects statistics on the SEDRIS objects that are used to compute metadata. The second traversal extracts the objects that will be used to construct the terrain database.

The metadata that is collected by the first traversal is used to set elevation grid spacing and also to construct the polygon attribute table. The type of object that will be extracted for this search is very specific. Location objects are extracted to calculate the grid post spacing. FACC attribute objects are used to build the polygon attribute table. Accordingly, a search rule is used that selects only these objects and ignores the intervening hierarchy.

The second traversal is programmed as a series of recursive one-level searches of the SEDRIS transmittal. This design allows the compiler to take advantage of any spatial organization that might exist. If a spatially organized union is found during one of the searches, those components of the union that do not overlap the search area are ignored. This provides some efficiency improvements when only a small part of the database is being processed.

It was originally intended that the object extraction routine would write object data to a buffer. The buffer would then be sorted so that the object data could be passed to the front-end processor in the correct order, with objects grouped by location. This scheme required multiple traversals of the SEDRIS transmittal in order to keep the size of the buffer small enough to fit in memory.

However, when it was determined that the compilation time would be proportional to the number of traversals of the SEDRIS transmittal, the object extraction routine was redesigned so that the object data would be written to a series of temporary files. The files act as buffers for the object data. Since they can be written simultaneously, only one traversal of the SEDRIS transmittal is necessary.

When a SEDRIS object is extracted, the compiler determines where it will be used and writes it to the appropriate temporary file. Objects that will be used for the elevation and soil grids are written to one file; abstract feature objects are written to another. Physical feature objects are written to several files, depending upon their geographic location. Objects may be written to more than one file if they will be used more than once.

Determining where an object will be used is part of the identification and classification process. This process is described in the next section.

4.2 Identifying and Classifying Objects

Traversing the SEDRIS hierarchy creates a linear list of SEDRIS objects. These objects must be tested to determine if they correspond to objects in the CTDB. If they do, they must be further tested to determine to which CTDB object they will be converted.
<table>
<thead>
<tr>
<th>CTDB Object</th>
<th>SEDRIS Object</th>
<th>FACC Code (for S1000 Implementation)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terrain Polygon</td>
<td>Polygon</td>
<td>DA030</td>
<td>FACC Attribute PAC (active) must not be set to 0</td>
</tr>
<tr>
<td>Building</td>
<td>Geometry Model Instance</td>
<td>AL015</td>
<td>Excludes Point Features</td>
</tr>
<tr>
<td>Bridge</td>
<td>Geometry Model Instance</td>
<td>AQ040</td>
<td>Excludes Point Features</td>
</tr>
<tr>
<td>Road</td>
<td>Linear Feature</td>
<td>AP000 or AP030</td>
<td>Excludes Polygons</td>
</tr>
<tr>
<td>River</td>
<td>Linear Feature</td>
<td>BH000, BH020, or BH140</td>
<td>Excludes Polygons</td>
</tr>
<tr>
<td>Tree or Tree line</td>
<td>Point Feature or Linear Feature</td>
<td>EC030</td>
<td></td>
</tr>
<tr>
<td>Tree Canopy</td>
<td>Linear Feature or Area Feature</td>
<td>EA040, EC015, or EE000</td>
<td>Contains both canopy roof and forest edge polygons. They are extracted together and separated during front-end processing.</td>
</tr>
</tbody>
</table>

Table 1. Object Classification

The first item to be tested in classifying objects is the SEDRIS object type. Generally only primitive geometry and primitive features need to be considered for conversion. In addition, not all SEDRIS object types can be converted to every CTDB object type. For example, only linear features can be converted to roads.

Screening by SEDRIS object type is also done so that some models may be expanded while others are not. A database may have tree models, which are not used by the CTDB. To avoid expanding these models, model instances of trees are rejected while their associated point features are accepted. Building and bridge models are expanded, so their model instances are accepted and their associated point features are rejected.

The Classification Data object is used in SEDRIS to define what the SEDRIS object represents in the synthetic environment. The classification data is a 5 alphanumeric character string that corresponds to a code in the Features and Attributes Coding Catalog (FACC) (see Section 0). Since different implementers might chose different FACC codes, a separate data file is provided to the compiler to define the mapping of FACC codes to CTDB object types. The data file provides flexibility in encoding the mapping information.

FACC attributes are also tested. Some polygons may be included in the data but are not part of the terrain because they have been superseded by other terrain. They are “inactive.” Inactive terrain polygons are identified by a FACC attribute. This FACC attribute is tested in order to reject these inactive polygons.

Table 1 lists the classification criteria used for the S1000 databases that were compiled. For each CTDB object, the SEDRIS object type, from which it is converted, and the identifying FACC code are listed.

4.3 Extracting Information about Objects

The CTDB needs several types of information about an object. The most important is its location. However, other data that describes characteristics of the object, such as the FACC code, FACC attribute code, and mobility index, are also needed. This data is stored in a variety of sources in SEDRIS.
The extraction of each type of data has been centralized within the compiler. Thus, all location information is extracted within a single routine, regardless of SEDRIS object type. The extraction of FACC codes, FACC attribute values, and mobility index are also centralized in separate routines. This centralization has the advantage that any changes in how a particular type of data item is extracted only needs to be made in one place.

### 4.4 Processing Objects

Once the data has been collected for an object and stored, the data must be converted to a form suitable for processing by the “back end” of the compiler. The back end also requires that the data be provided in the correct order. In particular, all the data that will be stored as features must be grouped in 500-meter squares called patches. The grouping is accomplished by storing the extracted data objects in an array and arranging the order of the objects to match the requirements of the back end.

When objects were classified, they were written to temporary files. The data that would be used for patches was written to one or more files depending upon its location. Each file corresponded to a set of patches that were laid out as a strip. This strip extends the width of the database and is four patches high. Any data needed by a patch in that strip is written to the file corresponding to that strip.

When the patches of a strip are to be processed, the data file corresponding to that strip is read. The objects in the file are then sorted first by the minimum y-value of each object. The objects whose minimum y-value shows that they are in the first row of patches are then sorted by their minimum x-value. This results in a list of objects that are loosely ordered by patch. The compiler processes the objects for a patch in ascending order within the array, and it stops when the minimum x-value of the current object indicates there are no more objects for that patch in the list.

### 5. Results

The compiler has been tested on two databases, both of which use the Lockheed-Martin S1000 format. The Bellevue database is small (10,000 x 10,000 meters) and is organized on a 125-meter grid. The Moba database is larger (24,000 x 24,000 meters) and is tinned. The results of the compilation are shown in the following table. The compilations were run on an SGI Indy with the debug option turned on. (Note that the use of the debug option will result in slower execution times for the compiler than might be seen in a production environment.) For comparison, compilation times using the earlier compiler, s1kprep, have been included. S1kprep compiles a CTDB directly from the S1000 data.

<table>
<thead>
<tr>
<th></th>
<th>Bellevue</th>
<th>Bellevue With s1kprep</th>
<th>Moba</th>
<th>Moba With s1kprep</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of CTDB Objects</td>
<td>21308</td>
<td>N/A</td>
<td>520121</td>
<td>N/A</td>
</tr>
<tr>
<td>Metadata Processing Time (minutes)</td>
<td>2.6</td>
<td>N/A</td>
<td>201.2</td>
<td>N/A</td>
</tr>
<tr>
<td>Object Extraction Time (minutes)</td>
<td>7.8</td>
<td>N/A</td>
<td>350.6</td>
<td>N/A</td>
</tr>
<tr>
<td>Total Compilation Time (minutes)</td>
<td>11.0</td>
<td>4.4</td>
<td>577.1</td>
<td>529.5</td>
</tr>
<tr>
<td>File Size (Mbytes)</td>
<td>0.294</td>
<td>0.284</td>
<td>9.03</td>
<td>8.70</td>
</tr>
</tbody>
</table>

Table 2. SEDRIS Compiler Performance Results

The time of the first traversal is considerably smaller than that of the second. The reason for this is that the second traversal extracts more data from the objects that it finds than does the first traversal. The first traversal reads the location coordinates and the FACC attribute labels from the attributes of the objects that it finds. The second traversal extracts information from component, link attribute, and associate objects of the object that it found. Searching for these additional objects accounts for the increased processing time.

The total compilation time for the SEDRIS compiler is longer than the compilation time using s1kprep. The increased time appears to be due to object creation and
deletion in the SEDRIS API. Whenever a search encounters a new object, memory space for that object must be allocated, and the object must be initialized. This effect is particularly pronounced when an association is searched. Because the hierarchy of the associate may not be in memory when the associate object is accessed, all the hierarchy objects must be created in order to access the associate object.

Based on the data from Bellevue and Moba, and assuming that the traversal algorithm is of type $O(n^x)$, the exponent $x$ can be calculated for each traversal. For the first traversal (metadata processing), the exponent $x$ is as follows.

$$x = \frac{\log(201.2/2.6)}{\log(520121/21308)} \approx 1.4$$

For the second traversal (extraction of objects), the exponent $x$ is as follows.

$$x = \frac{\log(350.6/7.8)}{\log(520121/21308)} = 1.2$$

The relative percentages of time taken by each phase of the compilation for the two databases are shown in the chart in Figure 2. The second traversal still takes the major portion of the compilation time as the size of the database increases from Bellevue to Moba. However, the relative inefficiency of the first traversal causes it to consume a greater proportion of the compilation time for the larger database. The time consumed by the front and back ends of the compiler is minor for both databases. This shows that most of the work in the compilation consists in extracting the data and putting it into the proper format.

![Figure 2. Relative time taken by each phase of compilation](image)

1.

6. References


Author Biographies

DR. VICTOR SKOWRONSKI is a Senior Member of the Technical Staff at TASC, where he investigates terrain representation for CGF systems. Prior to joining TASC in 1996, he did research in solid modeling at Rensselaer Polytechnic Institute, where he also earned a PhD in Computer Engineering. Victor
earned a M.E. and a B.E. in Electrical Engineering from Stevens Institute of Technology. He is a licensed Professional Engineer in New York and Massachusetts.

THOMAS STANZIONE is the manager of the Computer Generated Forces Section at TASC. He is the Program Manager for the DARPA Framework of Reusable Objects in the Synthetic environmenT (FROST) project. His interests include data representation for terrain reasoning and terrain database generation for simulation applications. Mr. Stanzione has a Masters of Science Degree in Photographic Science from the Rochester Institute of Technology.