Lessons Learned Developing a CDB Run Time Publisher

Douglas Price, Research Director; Robert Mingee, Software Architect
Presagis, USA
doug.price@presagis.com; robert.mingee@presagis.com

Abstract. Historically, synthetic environments have been generated off-line, often requiring unique, and sometimes proprietary, databases to be generated for each sub-system of a simulator. While this approach has been used successfully in many fielded systems, its disadvantages can include lengthy database production times, data duplication, and the introduction of correlation errors due to the variety of tools used to generate the content. The Common Database (CDB) initiative was developed for the United States Special Operations Command (USSOCOM) to facilitate the rapid creation of databases necessary to meet their mission rehearsal timelines. CDB defines a single database repository consisting entirely of open formats, which provides source level correlation for all components of the simulation. CDB also introduces the concept of a Run Time Publisher (RTP) that operates on these open formats to convert the data at run time into the form most optimally suited for each Client.

In spite of these benefits, CDB has only been deployed in a handful of simulators. This slow adoption may be due, in part, to a lack of software tools that read and write CDB databases. Unfortunately, for those software developers interested in adding CDB support to their tools, the specification itself, as one might expect, does very little to describe how one goes about developing a RTP, and currently there is no companion document that covers this topic. To that end, this paper catalogues the effort recently undertaken to add CDB support to the Vega Prime™ visualization software toolkit. The paper will describe the conceptual design of a RTP as outlined by the specification but will focus primarily on the design trade-offs required to augment existing database and run time constructs to work effectively with CDB concepts. An analysis of performance and resource management will also be presented.

1. INTRODUCTION

The requirements that simulator sub-systems place on the synthetic environments that they consume can vary widely. In an effort to achieve optimal fidelity and performance, each sub-system would ideally ingest only the data that it cares about, when it needs it, and in the formats it wants. Given these varying requirements, it is no wonder that many systems integrators have chosen to use unique, and often proprietary, data formats for each sub-system. While this approach has been used successfully in many fielded systems, it has some rather significant drawbacks.

First, the use of multiple data formats often requires the use of multiple tools and processes, which can lead to correlation issues. Second, data duplication creates configuration management issues as well as additional hardware storage requirements. Third, the time required to generate the content can be quite lengthy. Not only is additional processing time required to create multiple databases, but the nature of offline database generation means that even incremental changes to source data require the regeneration of all portions of the database affected by the change.

CDB was developed as part of the Army Special Operations Aviation Training and Rehearsal System – Fiscal Year 2004 (ASTARS-04) program for the 160th Special Operations Aviation Regiment (Airborne) (SOAR(A)) to enable the creation of databases that met their mission rehearsal timelines. Although it was initially aviation specific, the specification [1] was expanded to accommodate non-aviation users and was eventually adopted by USSOCOM as a database standard. Today, management of the standard has been transferred to Presagis™, who has taken on the responsibility of working with the community to modify and maintain the standard going forward.

Unlike other standards, CDB was designed to serve not only as a source database repository, but also as a run time format. As a repository, CDB is composed entirely of existing open standards, such as TIFF, sgi, and JPEG 2000 for raster data, Shapefile for vectors, OpenFlight™ for models, and XML for metadata. Zip files are also used to group models and their corresponding textures into pageable units. This single repository can be shared by all simulator sub-systems, thus eliminating data duplication and providing source level correlation. Since the content for CDB is generated at run time, even major database modifications can be seen in minutes, thus significantly reducing database construction time while still providing each sub-system with the data necessary to meet their individual requirements.

While several papers [2],[3] have been written citing the benefits of the CDB format, there exists very little documentation describing what is actually involved in the development of a RTP necessary to integrate CDB into existing software architectures. The objective of this paper is to address this need by documenting the design decisions and trade-offs that were made when adding CDB support to the Vega Prime visualization software toolkit.

2. CONCEPTUAL DESIGN

At a fundamental level, the CDB specification describes a way to organize the data necessary to create a synthetic environment of the entire world. In order to store this information in a way that allows for efficient search and retrieval, the specification organizes the data

into tiles, layers, and Levels of Detail (LOD). First, CDB sub-divides the content into tiles along latitude / longitude boundaries. Within these geographical tiles, or "geo-tiles", the data is then logically organized into distinct layers of information, as shown below.

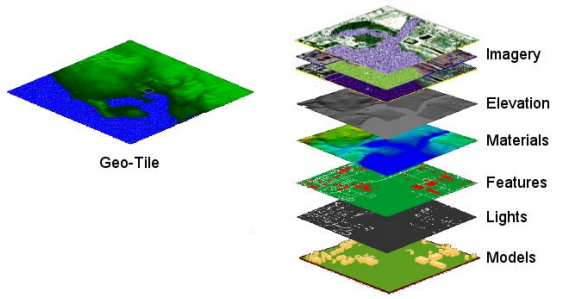

Figure 1: CDB Layers

So, for example, CDB separates the visual imagery from the elevation data, which is then separated from the feature data and so on. Since these layers are notionally independent of each other, changes to one layer can be made easily without impacting the others. Lastly, within each geographic layer, the data is further sub-divided into individual LODs. The specification defines 34 LODs using a quad tree structure that enables the creation of databases with near infinite (up to 13 micron) resolution. Because of the coverage, variety, and resolution of CDB, the amount of data required to represent a database can be quite large. However, this amount of data is generally much smaller than what would be required to store a database of equal content using traditional means.

In order to produce database content at run time, CDB introduces the concept of a RTP. The job of the RTP is to field requests from the Client, to read the source data from the CDB corresponding to the request, and to transform the data into the format(s) required by the Client. While this sort of run time data compilation may have been impossible in the past, advances in CPU and GPU technology have made it not only possible, but also viable for simulators in a variety of training applications.

The final piece of the CDB architecture is the Client itself. It is the Client's job to integrate with the existing sub-system software and to make requests to the RTP for data as needed. Technically speaking, it is possible for the Client to read the CDB directly, but, when integrating into existing systems, it is often easier and more efficient to use a RTP. The level of effort involved in this integration depends heavily on how similar the existing sub-system design is to the one put forth by CDB.

Putting these pieces together, we get a picture of what a conceptual CDB simulator might look like as shown in the figure below. Note that, while it is possible for multiple clients to share the same RTP, we have, for the sake of simplicity, only shown a single RTP per client in the figure.

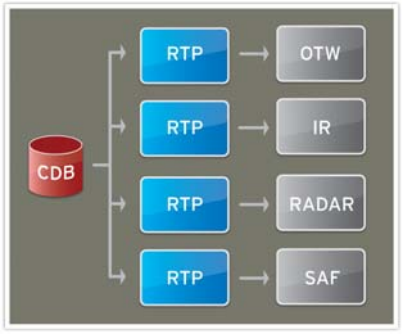

Figure 2: CDB Conceptual Design

3. VEGA PRIME DESIGN

3.1 Scope

Given the rich feature set that CDB provides, the amount of work required to build a fully compliant CDB RTP and Client would be quite sizeable. For our first release, we decided to keep things as simple as possible by narrowing our focus to only those features that we felt would allow us to deliver a single use case. We decided on the high flight use case, which we felt could be achieved by implementing regular meshed terrain overlaid with Visible Spectrum Terrain Imagery (VSTI), geo-specific and geo-typical point features, and integrated sites. While this falls far short of exercising the full CDB feature set, we believe this provided a good foundation for future CDB work, while still allowing us to provide a useful product offering to our customers.

3.2 Structure

As mentioned earlier, CDB sub-divides the world into geo-tiles, layers, and LODs. However, this approach produces tiles that become increasingly elongated as one moves towards the poles. In an effort to keep the area covered by the tiles more or less square, the specification assigns the geo-tiles to different zones where the width of the geo-tile in degrees varies based on the latitude of its southwest corner. So for example, between -50 and 50 degrees latitude geo-tiles are 1x1 degree, between 50 and 70 degrees latitude geo-tiles are 1x2 degrees, and so on.

For Vega Prime, the CDB structure posed some rather interesting challenges. Our geometry and imagery paging solutions, called Vega Prime LADBM™ (LADBM) and Vega Prime Virtual Texture™ (VT) respectively, both use the database description MetaFlight to define the structure of the database. MetaFlight allows the user to sub-divide the database coverage in any way they choose, so mimicking the quad tree structure imposed by CDB was rather straightforward. However, each dataset in MetaFlight can only reference a single grid structure. As a result, we had to represent each CDB layer with 11 MetaFlight datasets, one for each zone. While certainly less than ideal, the MetaFlight data structures are fairly light weight, resulting in minimal overhead.

In addition, while CDB and MetaFlight both maintain similar philosophies of dividing the database up into layers and LODs, the ways in which they address the individual files are different. CDB uses the latitude and longitude to identify the parent geo-tile and then uses LOD and a latitude and longitude index to identify the actual file within the geo-tile. In MetaFlight, which has traditionally been used to describe a single coverage area regardless of its size, file addressing is done simply using a level, row, and column index within the coverage area. So, in essence, the MetaFlight level, row, and column index was the same as the CDB LOD, latitude index, and longitude index, but MetaFlight had no construct that we could use to specify which geo-cell the level, row, and column was indexing into. To complicate matters, expanding the addressing to cover too many geo-cells required changing the integer data type used in the MetaFlight specification. Fortunately, the use of multiple MetaFlight files to specify the data structure for the CDB zones turned out to be a blessing as we were able to continue using level, row, and column file addressing within each zone. This not only kept us from having to make changes to the MetaFlight specification, but also allowed the existing LADBM and VT modules to work without any major modifications.

CDB defines all raster tiles, including elevation tiles, to be 1024x1024 in size. This is much larger and denser than the typical tile created using existing content creation methods, so we made the decision to allow for a configurable subdivision of CDB tiles into an even number of smaller tiles with the constraint that the size must be a power of 2, allowing the customer to decide at what granularity they wish to page and render tiles.

The same reasoning can be applied to cultural tiles, even though they are not regular mesh geometry. Limiting the geographic size of each tile results in fewer features per tile, meaning a smaller memory footprint, less processing time to load all features, and less network bandwidth used to transmit to the Client. It also allows paging to be more granular and thus more responsive at the edge of the paging range. This required managing some extra work in the RTP, but the benefits mentioned above far outweigh the extra overhead involved in subdividing CDB source tiles in the RTP.

3.3 Coordinate Systems

The specification requires that all information in the CDB be specified in a WGS-84 geodetic (latitude / longitude) coordinate system. The MetaFlight database description, on the other hand, actually defines two coordinate systems. The grid coordinate system is used to define the 2D layout of the tiles in the grid structure, while the data coordinate system is used to describe the 3D content within the tiles. So, for example, the grid structure could be defined in UTM coordinates while the content within the tiles might be specified in simple Cartesian (i.e. “flat earth”) coordinates. Of course, rendering a database of the whole world in a flat coordinate system would be of questionable value, so we defined the MetaFlight grid coordinate system as WGS-84 geodetic and the data coordinate system as Geocentric (3D Cartesian system with the origin at the center of the earth) so that we could render a realistic “round earth” view of the world. Since MetaFlight already supported both of these coordinate systems, this was a simple configuration step. Constraining the coordinate systems actually turned out to be a major simplification in the code since we no longer had to be concerned with all of the coordinate system permutations allowed by MetaFlight.

3.4 Levels of Detail

The CDB specification relies heavily on significant size, a concept introduced in Creator™ / OpenFlight to provide for more accurate LOD transitions. Unfortunately, Vega Prime only supported range-based LODs, so a small development effort was required to add this capability. With the scene graph capability in place, the next step was to make LADBM use significant size when creating its scene graph hierarchy. Since the CDB specification actually provides a table that describes how significant size values map to all of the LODs in the CDB, we were able to replace the values in the existing MetaFlight switch distance table with significant size values and then to make only a small modification to LADBM to tell it how to interpret these values. Once these changes were made, not only were our CDB tiles able to transition using significant size as expected, but all of our existing LOD functionality, such as LOD scaling, stress management, and multi-sample transitions, worked without further modification. As mentioned previously, significant size provides for much more accurate switching as it accounts for channel resolution, which is particularly important for culture. For terrain, however, the concept of significant size is less well-suited, so we decided to stick with a traditional range-based approach for terrain switching. The specification does not require that switching be done in a specific manner but merely provides significant size values appropriate for the various LODs.

3.5 Terrain

One of our primary objectives with this project was to use the existing architecture of Vega Prime as much as possible. With respect to geometry paging, this meant trying not to disturb the rather sizeable code base that is LADBM. As mentioned earlier, using the existing MetaFlight specification without modification went a long way towards achieving this goal, but we still needed a way for the Vega Prime Client to communicate with the RTP, ideally without having to make the rest of the LADBM code CDB aware. We were able to achieve this goal by creating a new geometry loader plug-in. In Vega Prime, when a request is made to load a file, the file extension is used to find the corresponding loader that loads the data and returns the result. So, by creating a new file extension that we specified using the MetaFlight file name pattern, we were able to use the existing LADBM paging logic to make requests to our new loader that would then send a request to the RTP and return the resulting tile to LADBM for processing.

On the RTP side, the regular spacing of the terrain made it easy to determine the geodetic position and texture coordinates of each vertex. When a request was received from the Client, the corresponding elevation tiles were read, if necessary, and the raster was queried to determine both the vertex elevation and normal. The final vertex position was then converted to geocentric and was localized to avoid floating point precision issues. The use of regular mesh also made it easier to adopt the TerraVista™ SmartMesh [4] approach to eliminate cracks between adjacent levels of the terrain. The tile geometry was then converted to an in-memory representation of VSB (Vega Scene Graph Binary), the internal binary file format in Vega Prime, and sent to the Client.

3.6 Culture

The approach of the Client with respect to culture was almost identical to that used for terrain because, to LADBM, both terrain and culture represent pieces of geometry that need to be paged and displayed. As a result, adding support for culture to the Client took almost no effort. In fact, we were even able to use the same loader for both purposes. The RTP, however, does need to make a distinction between terrain and culture so that it knows what type of geometry to create. To achieve this, we simply passed additional data to the loader so that it could make the appropriate calls to the RTP.

When the RTP receives a request from the Client, it starts by reading the corresponding vector data from the CDB. The vector attributes contain information about the feature, including its FACC, FSC, and model name, that can be used to determine the actual OpenFlight model file name. The resulting OpenFlight hierarchy for each feature is attached to the tile using a transformation based on the point's location, heading, and scale, as well as on an altitude determined by either an absolute value from the vector data or by intersecting the elevation raster at the feature location. This potentially large hierarchy is then optimized, converted to VSB format, and sent to the Client.

3.6.1 Light Points

In addition to models, the geo-specific cultural dataset also contains 2 sub-datasets for light points. One contains airfield lighting systems, and the other contains general wide-area cultural lighting. The vector data contains a type for each light point that can be used in conjunction with various standards to determine the visual behavior of the light in terms of blinking, color, directionality, visible range, etc. The CDB specification defines a hierarchical set of types (where more general types can contain sets of more specific types). We chose to use an XML file to provide a mapping between the light type and the light animation / appearance attributes in Vega Prime in order to facilitate tuning the behavior of lights on a per database basis. This is something commonly done with traditional databases to meet the desires of different customers with distinct training needs.

3.6.2 Polygonal Insets

Given the high flight use case that we adopted for the initial release, we obviously had a need to visualize airfields. We also had access to a large library of airfields already modeled as OpenFlight models. In order to facilitate reuse, we adopted the concept of geo-specific models so that, beyond a certain size, models are considered separately as insets and thereby are loaded separately from smaller geo-specific models and at different ranges.

These insets include the surrounding terrain because the terrain and flat surfaces, such as runways, of an airfield must be designed carefully so that distracting artifacts like z-fighting do not occur. The most efficient approach for rendering these insets would be to "cut away" the terrain under them and then stitch the edges for a seamless appearance. However, this was considered overly complex for the first release. Instead, we opted to model a downward-angled skirt around the perimeter of the inset in conjunction with a lowering of the elevation underneath the inset to prevent z-fighting between the elevation-generated terrain and the terrain modeled as part of the inset. This is illustrated in Figure 3 below.

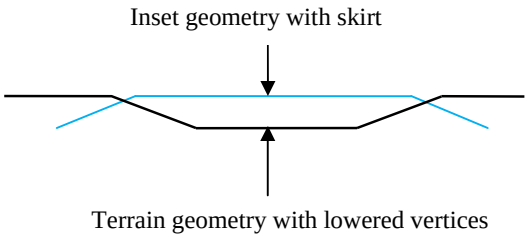

Figure 3: Polygonal Inset Design

In all other respects, these insets are handled in a fashion identical to all other geo-specific cultural features.

Eventually, the terrain (and/or rooftops) of these insets will be flagged so that the Client will apply VT to those polygons. This removes the need for applying textures at modeling time and creates a more seamless match with the surrounding terrain. It also allows for the textures of an inset to change along with the rest of the terrain when imagery is republished. Lastly, it allows polygonal insets to leverage all of the efficiencies of the VT approach. For the first release, however, we decided to continue modeling inset geometry with geo-typical textures since creating an approach that didn't require potentially time-consuming offline modification of the inset models was deemed to be too large of a development effort. For the future, the goal is to provide a run time based texture replacement solution that will require minimal involvement by the user.

3.7 Imagery

VT [5] uses a set of imagery organized on disk in the same manner as a single geo-tile in CDB. However, VT expects a single "mip-map stack" for the entire database, not for each geo-tile. A VT carries a reasonable amount of overhead in Vega Prime, so it

would not be feasible to simply create a VT for each geo-tile. Instead, we chose to decouple the organization of the imagery data in the CDB from the VT coverage in the Client, which is a movable square coverage sized by integral powers of 2 along each side. As mentioned previously, each zone is represented by a separate terrain dataset. In order to match this organization and to reduce the complexity of the imagery loader, we chose to create a VT for each zone as well. The Client manages which VTs should be actively loading image data based on the location of the observer.

The existing VT implementation used a single texture coordinate space across the coverage of the whole database to manage textures and their application to terrain tiles. This inherently limits the size of databases using VT because texture coordinates are limited to single-precision floats. Using traditional database construction techniques, the terrain was published with those texture coordinates already set and the run time could simply use them. Obviously, this strategy breaks down when the coverage becomes movable because the texture coordinates for a terrain tile change when the coverage changes. Instead, we chose to use the geodetic tile extents to manage both the creation of textures and the application of those textures to tiles since those values are fixed. In other words, the different levels of VT now store an extent in degrees for each texture, and the extents of these textures can be compared against the extents of each tile with the goal of finding the highest detail texture that completely covers the tile.

Apart from that change, much of the infrastructure of VT was reused in the Client. However, the loading of imagery had to be replaced since there is no fixed grid structure corresponding to a set of image tiles on disk because each geo-tile in CDB is a separate mip-map stack. The situation is further complicated by the fact that the coverage is movable. We were still able to use Vega Prime's plug-in loader architecture to create a new VT texel loader; however, instead of being given a level, column, and row, the new loader is given a geodetic coverage and target resolution parameters that it uses to determine which imagery tiles to query from the RTP.

The RTP simply receives the request for a given tile, and, if that tile is present in the CDB, it loads it. Then, the RTP converts it to DXT1 compressed because our VT implementation is set up to handle that format. This is very efficient because it can be used directly by most modern graphics cards, saving both system and video texture memory. If there is no file present, the RTP simply returns nothing.

VT allows for different techniques to be created that facilitate the blending of multiple data sources for advanced effects, such as bump mapping or light mapping. The CDB specification, however, does not provide for the publishing of a bump map on terrain. While one could be generated dynamically from the elevation in the RTP, we decided not to attempt this for the first release. Instead, we simply use per-vertex normals, which produce better shading on the regular mesh of CDB terrain than on the TIN common to traditional database publishing tools. CDB does provide for light maps, but our VT implementation does not currently have a light map technique, so we decided not to attempt this for the first release either.

3.8 Resource Management

The potentially large amount of data being handled by the CDB architecture makes it important to carefully manage resources, particularly memory usage. Vega Prime contains a number of built-in mechanisms for this--such as palettes for storing textures, shaders, and other state information--that can be leveraged in the Client to reduce the memory usage for geometry. This is particularly important for culture, which is much more heterogeneous than terrain since all terrain tiles share the same state to facilitate VT. In addition to saving memory, this sharing also reduces the number of state changes, which improves rendering efficiency. As mentioned in the imagery section, the use of DXT compressed textures for all terrain and cultural imagery also reduces video memory consumption and sub-loading times.

Another technique for reducing memory usage is to load only a level of texture appropriate to a given cultural tile LOD. For instance, if a tile is a lower LOD that will only be seen from a distance, there is no point in loading high detail textures for the models because only the lower mip-maps for those textures will be seen. The CDB specification calls for texture mip-maps to be stored independently, which makes this process easy. The desired level of image can be read directly from the CDB, and then the mip-maps can be either read or generated. Textures are also given LOD numbers just like all data in CDB, and the RTP has an option to select the number of levels above the vector data LOD when requesting textures. In most cases, requesting the same LOD of texture as the vector data produces results that are too low in detail, but the user can balance memory usage with visual quality.

All datasets can be configured in the Client to specify the LOD at which to start loading data, a feature that can be useful when you know that you will be working with very small far clip distances. You can also specify the LOD at which to stop loading data in order to limit the content to the level that the Client machine is capable of handling. For example, a Client running on a laptop will likely want to specify a much lower LOD than a Client running on a high end workstation. For the first release, this selection is static, meaning that the application must be restarted for it to take effect, but, in the future, it will be dynamic.

3.9 Performance

A number of techniques were implemented to improve rendering performance, including the use of texture atlases. Geo-specific and geo-typical culture tiles reference a number of different textures for their models, which results in a number of state changes when rendering the tile. By combining the textures into a single large atlas, the whole tile can be rendered using

a single texture, which greatly reduces state changes and improves rendering performance. In order to make use of texture atlases--which don't allow repetitions--the RTP uses the texture coordinates of the geometry to detect repeating patterns and then either explicitly repeats the image in the atlas or excludes it altogether. Despite these limitations, the use of texture atlases provided significant benefits in most cases.

Sometimes, as in the case of caching built tiles, memory usage and performance are at odds with each other. Once the effort has been performed to generate a tile, particularly a cultural tile with a number of models, we deemed it prudent to cache those tiles for a certain period of time, level of memory usage, or until the Client specifically "flushed" them. That way, if you have multiple clients requesting the same data or a single Client frequently revisiting an area, you can fulfill requests much faster by caching the fully built tile.

In order to improve throughput, one technique we used was to have the Client give the RTP "hints" about when to build a tile in anticipation of loading that tile. In other words, if you page in tiles within a given range, the Client requests that the RTP start building tiles in a slightly larger range. By the time the Client is ready to load the tile, the RTP may have already built it and have it ready to send.

3.10 Future Work

As mentioned previously, the feature set for the initial release focused on only a single use case, and there are many features on the roadmap for the future. A number of these are modules that are already a part of Vega Prime but that are not yet currently integrated with the CDB solution.

The most important of these is support for wavebands other than the visual spectrum, and that integration effort is well underway. This feature could pose very interesting challenges because of the compute-intensive nature of generating textures for use in other wavebands, something that is done as an offline publishing step with a traditional database. New technologies, such as CUDA, are being investigated for their viability in helping to solve this problem. Support for other wavebands in culture may require changes to CDB to support the material classification of models at the texel level as only polygonal classification is available today.

Another important feature is support for marine simulation. Our marine solution includes both the use of shaders at a distance to give the impression of water, as well as actual dynamic ocean surface geometry at close range. The former should be fairly straightforward to implement as it will only require RTP changes to attribute the water geometry appropriately, but the latter could be quite challenging as it will represent a fairly fundamental change to the way the water geometry is managed and integrated with land. For both solutions, coastline tiles may provide some interesting challenges when applying marine techniques only to the portions of the tile that represent water.

Vega Prime currently supports the instancing or batching of frequently-used models for improved rendering performance. This would be ideal for improving the performance of geo-typical culture loaded from the CDB, but we decided to wait to implement this until a future release.

Vega Prime also supports the application of micro-texture for improving the appearance of lower resolution imagery at very close range. Since material classification is integral to CDB, the application of micro-texture, or hyper-texture as it is called in Vega Prime, could be extended for use on different textures based on their material classification.

Besides the integration of existing modules, we also plan to add support for other data within the CDB that is not currently loaded by our RTP. These include linear and aerial features in the vector cultural data, as well as light maps for terrain.

4. CONCLUSIONS

While CDB offers many benefits, the breadth of the specification makes the development of a fully CDB-compliant simulator a sizeable undertaking. Fortunately, CDB is organized in such a way that it makes it fairly straightforward to implement some capabilities while ignoring others. By targeting only those features that addressed a specific use case, we were able to have a basic prototype up and running fairly quickly and then spent the rest of our development cycle optimizing these features. The result was that, during a standard release cycle, we were able to deliver a CDB solution that addresses a specific set of customer requirements while giving us a solid foundation on which to add additional capabilities in the future.

5. REFERENCES

1. (2007) "Common Database (CDB) Specification for USSOCOM, Rev. 3.0" U.S. Army Program Executive Office for Simulation, Training, and Instrumentation (PEO STRI).
2. Simons, R. & Lagace, M. (2004) "The Common Database – An All Encompassing Environmental Database for Networking Special Operations Simulation" 2004 Image Conference Proceedings.
3. Lalonde, B. (2005) "CDB – A Common Environment for Real-time Simulation" 2005 SimTecT Conference Proceedings.
4. Willis, L. (1998) "Who Says You Can't Teach an Old LOD New Tricks" 1998 Image Conference Proceedings.
5. Ephanov, A. & Coleman, C. (2006) "Virtual Texture: A Large Area Raster Resource for the GPU" 2006 I/ITSEC Conference Proceedings.