Terrain Deformation Software and osgTDS

User's Manual

Version 1.0
April 7, 2007

CGSD Real Time Terrain™ Deformation Tools User's Manual

The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by CGSD Corporation. CGSD Corporation assumes no responsibility or liability for any error or inaccuracy that may appear in this document. CGSD Corporation assumes no responsibility or liability for any damage that may be caused by using the software.

Many of the designations used by manufacturers and sellers to designate their products are claimed as trademarks. Where the designations appear in this manual, and the author was aware of a trademark claim, the designations have ben printed with initial capital letters or in all capitals.

Contents

Preface

This User's Manual describes an algorithm for real-time terrain deformation and its implementation in the OpenSceneGraph API.

Chapter 1, **Introduction to the Terrain Deformation Software,** contains a description of the real-time terrain deformation algorithm. It's written for a software developer or database modeler with some knowledge of mathematics and terrain database modeling. After reading this chapter, a developer or modeler should have a firm understanding of how the algorithm works, both at the concept and formula levels.

Chapter 2, **osgTDS—TDS in OpenSceneGraph,** contains information about the OpenSceneGraph implementation of the algorithm called osgTDS. It's written for a system administrator or software developer with some knowledge of system configuration and software development. After reading this chapter, a system administrator or software developer should know how to install osgTDS on a run-time platform, so that OSG-based applications can use it.

Chapter 3, **osgTDS Code Internals,** contains detailed information about the design and code structure of osgTDS. It's written for a software developer with a firm knowledge of software development, C++, and the OpenSceneGraph API. After reading this chapter, a software developer should be familiar enough with the osgTDS code base to add features or resolve issues.

Chapter 4, **Issue Tracking,** and chapter 5, **Bibliography,** contain issue tracking and bibliographic information, respectively.

1 Introduction to the Terrain Deformation Software

Flight simulators use large-scale terrain databases and building (or target) models to simulate a real-world environment. For an accurate and visually acceptable simulation, the elevation data of the terrain, as well as the targets, must be both accurate and precisely correlated.

Target elevations are often acquired with extreme accuracy. The elevation data used to create the corresponding terrain databases, however, is often inadequately sampled or based on outdated measurements. As a result, mismatches occur between target elevations and the corresponding terrain elevation at the target location. These mismatches result in targets that appear to float above or sink into the terrain, as Figure 1-1 illustrates.

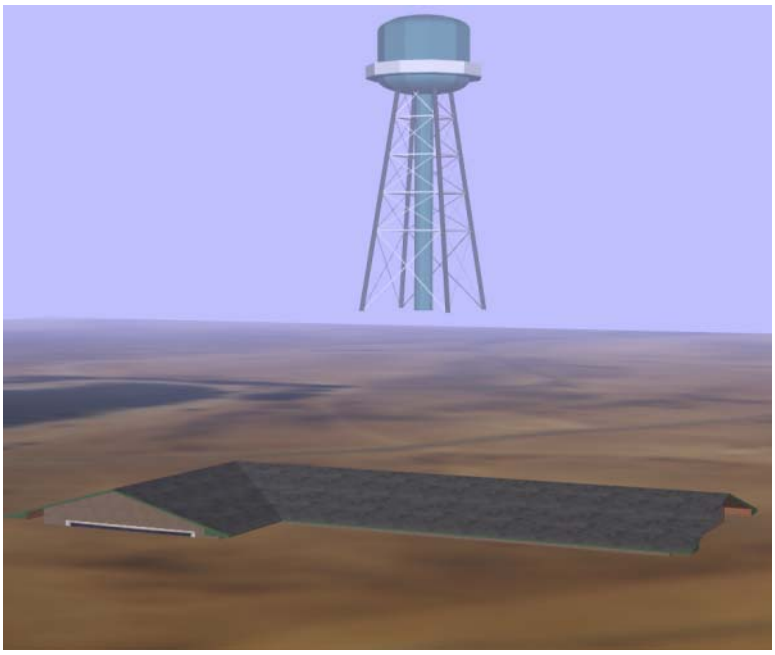

Figure 1-1
Target Elevation Mismatches
This image shows a terrain database with two targets. The target elevations don't match the terrain elevation data, causing visual aberrations. The water tower appears to float above the terrain, and the house appears to sink into the terrain.

These visual aberrations are often minimized by adjusting the target elevation to correspond to the terrain elevation, which compromises the accuracy of the target measurement. Alternatively, the terrain database can be modified offline using modeling software, which is a time-consuming and labor-intensive process.

The Terrain Deformation Software (TDS) is a real-time terrain deformation algorithm that modifies terrain databases to conform to newly acquired target elevation measurements. TDS modifies the terrain during database load to seamlessly correspond to target elevations. Modifications to the terrain database are local to

the target area and conform naturally to existing terrain data. This process preserves target location accuracy while eliminating visual aberrations due to elevation data mismatches with no offline terrain database modification expense. The algorithm was published in the IMAGE 2006 paper, "Dynamic Terrain Modification Using A Correction Algorithm," by Roy Latham and Donald Burns. TDS was developed by CGSD Corporation with funding from the US Air Force (SBIR topic number AF04-064).

Figure 1-2 shows the same scene as Figure 1-1 after processing by TDS.

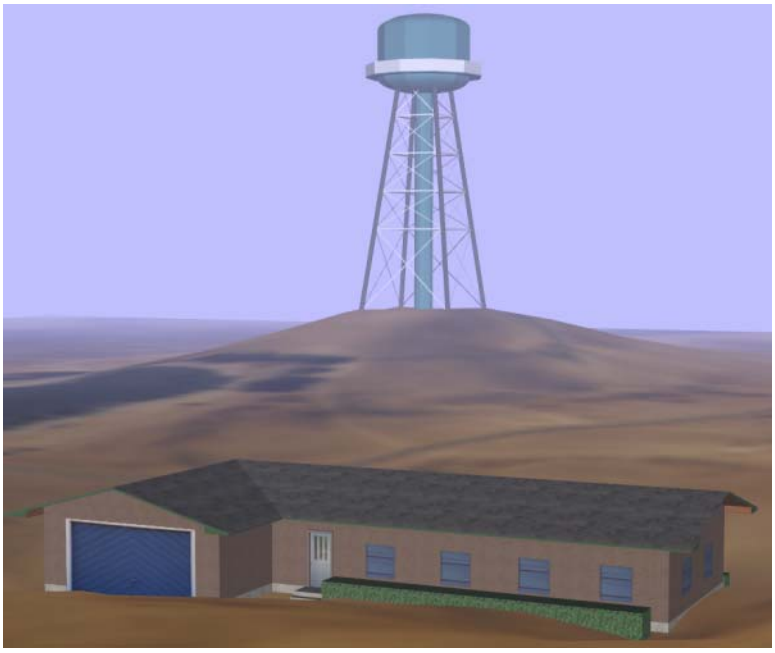

Figure 1-2
Terrain Database Processed by TDS
TDS modifies the terrain database at runtime to match target elevations. In this scene, a hill was created under the water tower, and a depression was formed under the house.

Targets are typically polygonal features representing buildings. However, the TDS algorithm allows point, line, and polygon targets. This flexibility allows TDS to modify terrain using a wide range of input data.

osgTDS is an open source implementation of TDS using OpenSceneGraph (OSG). Any OSG-based simulation application can use TDS by downloading and installing the osgTDS plugin. This open source implementation of TDS also serves as a reference source for other implementations.

1.1 Overview of the TDS Algorithm

To deform the terrain, TDS defines a correction function in such a way that, when TDS adds the correction function to every vertex in the original terrain database, the resulting elevations will conform exactly to the input target elevations, and will vary naturally between targets.

TDS applies the correction function to each vertex in the terrain database. If the resulting corrected triangle fails to accurately represent the corrected surface, TDS subdivides the triangle until it obtains an accurate representation.

The following sections describe the TDS algorithm in greater detail.

1.1.1 The Correction Function

To define the correction function, TDS triangularizes the correction function space with respect to the target vertices. TDS creates the triangulation from the target vertices and a set of vertices representing the extent of the correction function space. Figure 1-3 illustrates an example consisting of three target vertices.

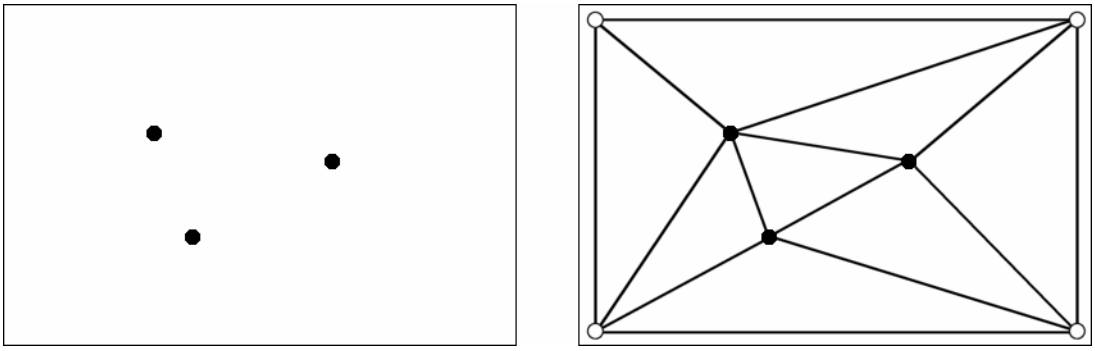

Figure 1-3
Triangularizing the Correction Function Space
Given three point targets, shown on the left, TDS creates a triangular mesh over the correction function space, as shown on the right. The mesh consists of the three target vertices plus four vertices that represent the correction function space extent.

The correction function is defined over the same x,y space as the terrain database, so the vertices representing the extent of the correction function space are the corners of the terrain database. These vertices need only represent the maximum area to correct, however, which is typically a subset of the entire terrain database.

The resulting triangulation of these vertices is called the correction space mesh, or CSM. TDS defines the correction function for each vertex in the CSM as the difference between the vertex elevation and the terrain elevation, as expressed in the following equation:

$$C(x_i,y_i) = a_i = z_i - T(x_i,y_i)$$

z_i is the elevation of the vertex located at x_i,y_i, and $T(x_i,y_i)$ is the elevation at x_i,y_i in the original terrain database. For a target vertex, z_i is the elevation of that vertex. For a vertex representing the extent of the correction function space, $z_i = T(x_i,y_i)$, such that $C(x_i,y_i)$ evaluates to 0 for that vertex.

The Influence Function

Within each CSM triangle, TDS defines the correction function based on the correction values at the triangle vertices. One possible correction function is linear interpolation of those values. However, for large corrections, linear interpolation produces long and unnatural features and other abrupt changes. To soften the effect of the correction, TDS creates natural hills or valleys with an influence function, which is defined as follows:

$$f_i(x,y) = a_i * \exp(d^2/k^2a_i^2)$$

In this equation, $d^2 = (x-x_i)^2+(y-y_i)^2$, and k is a slope constant, which is nominally 4. Increasing k makes the hills and valleys more gradual.

This influence function implements a Gaussian filter. Different influence functions produce different results. For example, influence functions can be designed to create craters or pillars in the terrain database.

Influence Function Weights

Given an influence function $f(x,y)$, TDS defines the correction function for any point within a CSM triangle as a weighted sum of the influence function at the triangle vertices.

$$C(x,y) = w_1f_1(x,y) + w_2f_2(x,y) + w_3f_3(x,y)$$

In this equation, x,y is within the triangle with vertices $[t_1,t_2,t_3]$. The weight values w_1,w_2,w_3 have the following properties:

- At a triangle vertex, the influence function weights of the other two vertices must be zero.
- Along a triangle edge, the influence function weight of the third vertex must be zero.

Given a triangle *ABC* and a point P within this triangle as Figure 1-4 shows, TDS computes the weights w_1,w_2,w_3 as follows:

w_1 = 1 - (length(A,P) / length(A,I_1))
w_2 = 1 - (length(B,P) / length(B,I_2))
w_3 = 1 - (length(C,P) / length(C,I_3))

In these equations, I_1 is the intersection between lines AP and BC; I_2 is the intersection between lines BP and CA; and I_3 is the intersection between lines CP and AB.

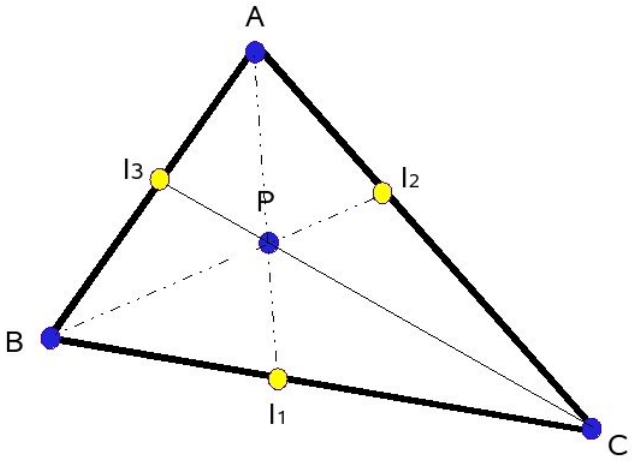

Figure 1-4
Computation of Influence Function Weights
This figure illustrates the values TDS uses to compute the influence function weights at each vertex of the given CSM triangle.

1.1.2 Terrain Processing

After creating the CSM and defining the correction function, TDS processes the terrain database and applies the correction function to each terrain vertex. If a corrected triangle does not accurately represent the corrected surface, TDS subdivides it into smaller triangles to create a better approximation.

TDS tests a terrain triangle by evaluating the following equation for each of the triangle's three component edges:

$$|[C(x_i,y_i) - C(x_j,y_j)]/2 - C([x_i-x_j]/2, [y_i-y_j]/2)| < \varepsilon_T$$

In this equation, x_i,y_i and x_j,y_j are two vertices of a triangle edge, and ε_T is a constant specifying the required accuracy of the triangle fit, nominally 0.5 meters. In simple terms, this equation computes the difference between the correction function value at the midpoint of the triangle edge and the average of the correction function values of the endpoints, and compares that difference against a constant.

If the equation fails (evaluates to false) for any of the three triangle edges, TDS subdivides the triangle into four component triangles. TDS repeats the process on the resulting triangles, stopping only when the equation above evaluates to true for all edges of all terrain triangles.

1.1.3 Special Cases

TDS uses the component vertices of the base of polygonal targets to define the CSM. However, the terrain within the hull of these vertices must be flat. To handle this special case, C(*x,y*) for vertices within the hull of a polygonal target evaluates to the elevation of the base of that target.

1.1.4 Optimizations

Only terrain triangles that intersect the CSM need to be processed. For this reason, the CSM should be as small as possible. Given a single point target *t*, the CSM must be large enough to encompass all points *p,* such that $|p - t| < z_v k$.

1.2 Illustration of the TDS Process

This section provides step by step illustrations of the TDS algorithm.

Figure 1-5 shows an unmodified terrain database with three correctly-positioned target models. The target elevations don't correspond to the terrain elevations. As a result, the targets float above the terrain.

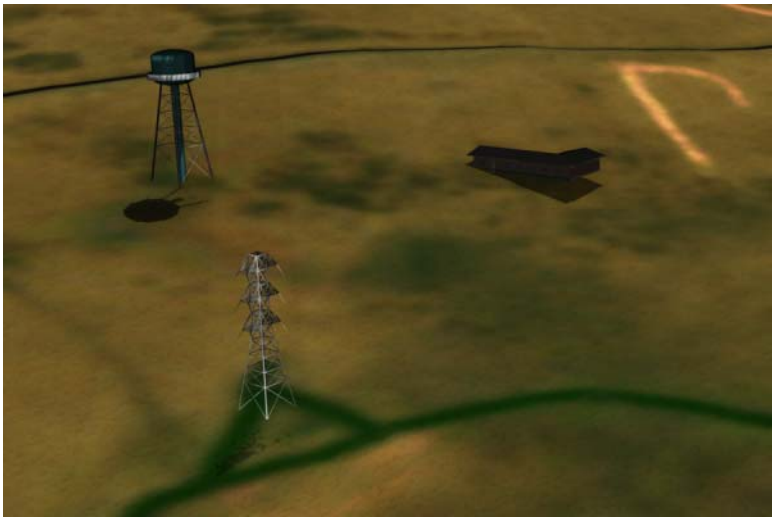

Figure 1-5
Unmodified Terrain
Before any TDS processing, the targets float above the unmodified terrain.

In Figure 1-6, the algorithm has added the bottom vertices of the targets to the terrain database. This places the targets in contact with the terrain, but the terrain modification is unnatural. The algorithm has not yet modified the terrain to fit the correction function.

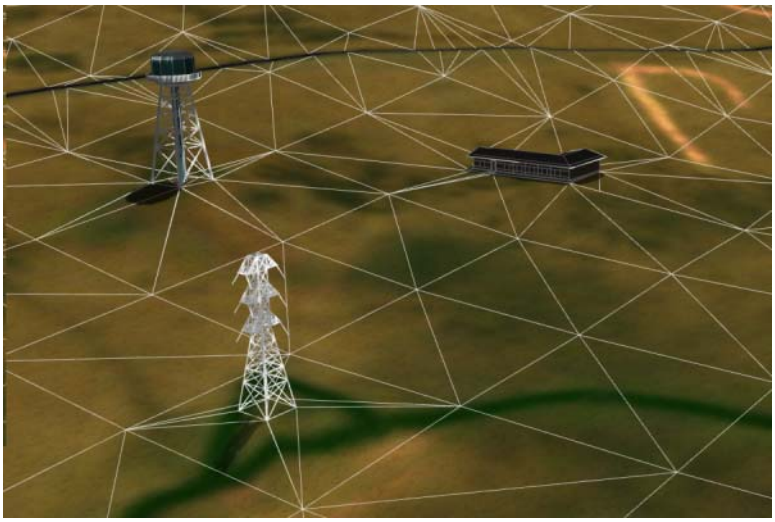

Figure 1-6
Target Vertices Added to Database
The database has been modified so that it contains the target vertices. Terrain triangles are outlined in white for clarity.

Figure 1-7 shows the database after the algorithm adds the correction value to the existing terrain elevations. The correction is small, so the difference between Figure 1-6 and Figure 1-7 is subtle. The terrain deformation is still unnatural because the algorithm hasn't performed any subdivision.

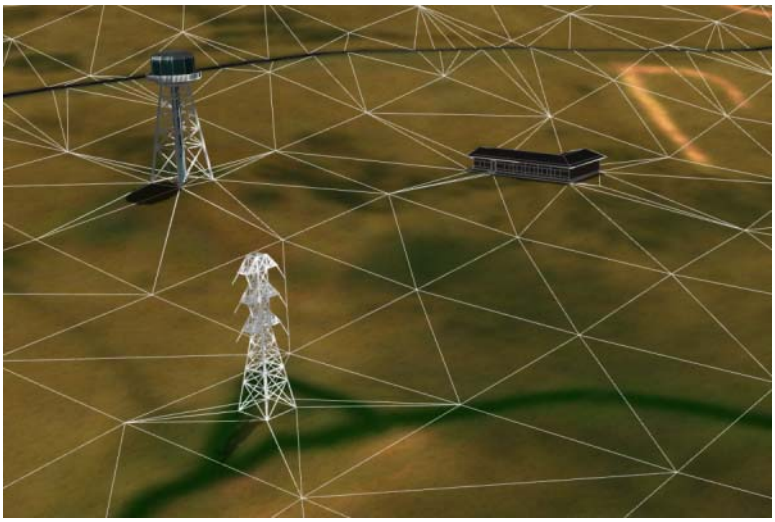

Figure 1-7
Correction Value Added to Existing Vertices
This figure shows the database after the algorithm adds the correction value to the existing terrain vertices. Terrain triangles are outlined in white for clarity.

Figure 1-8 shows the database after the algorithm has subdivided the existing terrain triangles to maximize fit to the correction function. This step adds several new triangles and vertices to the terrain database. After subdivision, the terrain deformations appear natural. This same image is reproduced as Figure 1-9 with triangle outlines removed.

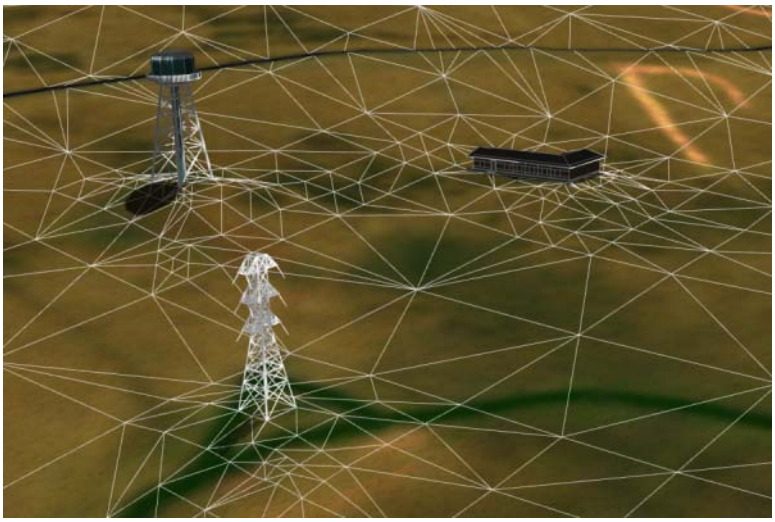

Figure 1-8
Subdivided Terrain
This image shows the result of the triangle subdivision step. The algorithm assigns elevation values to new vertices that maximize triangle fit to the correction function. Terrain triangles are outlined in white for clarity.

Figure 1-9
The Final Result
This is the same as the image in Figure 1-8 with triangle outlining removed. It shows how the database appears after the TDS algorithm has completely processed the terrain to fit the targets.

2 osgTDS—TDS in OpenSceneGraph

The TDS algorithm is implemented using OSG. This implementation allows immediate access to TDS for existing OSG-based applications, and serves as a reference source for future implementations.

OSG is an open source, cross-platform, scene graph library. The OSG v1.2 core is composed of three interdependent libraries:

- The osg library contains support for spatially organizing geometry and state controlling its appearance. It's composed of scene graph and related support classes.
- The osgUtil library contains classes for operating on the scene graph, which perform such operations as culling and optimization.
- The osgDB library contains visual database support, including mechanisms for reading 3D model and 2D image files, as well as other data files.

Figure 2-1 illustrates OSG's architecture.

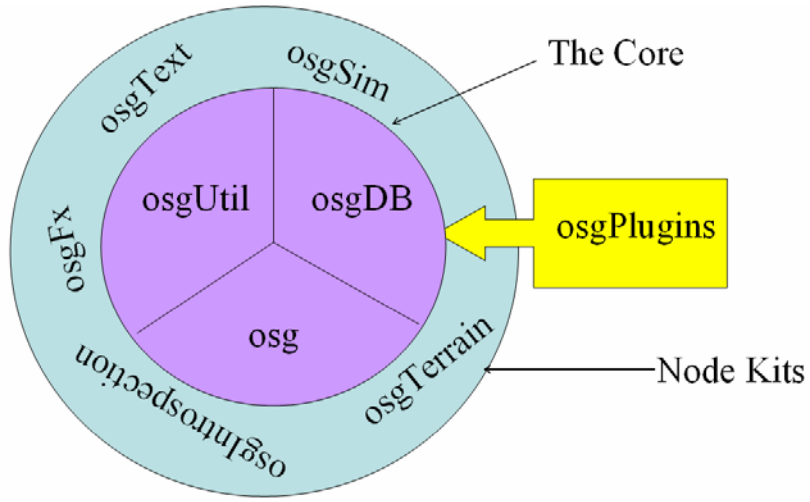

Figure 2-1
OpenSceneGraph Architecture
OSG's core is composed of three libraries: osg, osgUtil, and osgDB. NodeKit libraries build upon the core library functionality to provide additional higher-level features. The osgDB library provides access to the osgPlugin libraries for file I/O.

The osgDB library supports a large variety of file formats using a plugin architecture. To load a file of any particular type, you need to have the plugin for that file type installed on your system in a location where OSG can find it.

In OSG, TDS is implemented as an osgDB plugin. At runtime, this plugin loads a configuration file that specifies targets and information about the terrain database. As your application loads the terrain database, the TDS plugin modifies the terrain at load time based on information in the configuration file.

For more information on OSG, visit the following Web site:

`http://www.openscenegraph.org`

Also, refer to the book *OpenSceneGraph Quick Start Guide*. This book is modified as OSG continues to evolve. Information on the latest version is available at the following Web site.

`http://www.skew-matrix.com/OSGQSG.html`

2.1 Using osgTDS in Your Application

The following steps describe how to incorporate osgTDS into your OSG-based application for real time terrain deformation:

1. Install the osgTDS plugin on your system in a location where OSG can find it. If the osgTDS plugin isn't available in a binary form for your runtime platform, you need to build it from source code first. For information on this step, see 2.1.1 Obtaining the TDS Plugin.
2. Use a text editor to create a .TDS file containing information about your terrain database and target(s). For more information on this step, see 2.1.2 The .TDS File Format.
3. Before loading your terrain database, use the osgDB to load the .TDS file you created in step 2. After loading the .TDS file, osgTDS loads and returns to your application a scene graph containing all visible targets.
4. Use the osgDB to load your terrain database. osgTDS loads and processes each part of your database and returns to your application a scene graph containing the results of the deformation process.

The following sections describe obtaining the osgTDS plugin and creating the .TDS file. The section, **2.1.3 A Simple osgTDS Example,** provides an example of loading the .TDS file and the terrain database.

2.1.1 Obtaining the TDS Plugin

Precompiled versions of the osgTDS plugin are available for download for a limited number of platforms. Source code for the plugin is also available through anonymous CVS.

Downloading Binaries

This section tells you how to download and install precompiled osgTDS binaries for Microsoft Windows. The binary package was built using the Visual Studio .NET 2003 compiler and linked against OSG v1.2.

Download Microsoft Windows binaries from the osgTDS Web site:

`http://www.andesengineering.com/Projects/TDS/`

Unzip the binary package to any directory and execute the setup.exe file. After installation, manually add the installation directory to your PATH environment variable. The installation directory defaults to `C:\Program Files\osgTDS`.

To create osgTDS binaries for other platforms, you'll need to download the osgTDS source code and build the binary. The following section covers this topic.

Obtaining and Building the Source Code

If you have obtained osgTDS binaries for your platform and you aren't interested in reviewing the source code, skip this section and proceed directly to the following section, **2.1.2 The .TDS File Format.**

If osgTDS binaries aren't available for your platform, you'll need to download the osgTDS source code and build your own binaries. You might also want to download and review the source code if you intend to implement the TDS algorithm in another rendering environment.

To build osgTDS from source, you must have an OpenSceneGraph source tree on your build system. You can find Instructions for obtaining the OSG source code at the following Web site:

```
http://www.openscenegraph.org
```

Download the osgTDS source code using anonymous CVS:

```
cvs -d :pserver:cvsguest@andesengineering.com:/cvs/osg co osgTDS
```

Building and Installing for Linux Platforms

Linux builds have two prerequisites: DWMake and libXML v2.

DWMake

On Linux, osgTDS requires the DWMake Makefile definitions. (DWMake is not required to build under Windows.) Download DWMake via anonymous CVS:

```
cvs -d :pserver:cvsguest@andesengineering.com:/cvs/DWMake co DWMake
```

Follow the instructions in the README file to configure DWMake.

libXML v2

osgTDS requires libXML v2 for XML parsing support. (For Windows platforms, libXML v2 Windows binaries are included in the source tree.) Use your Linux software package installer to download and install the libxml2 development package.

Building and Installing osgTDS

After you check out osgTDS and DWMake from CVS and install the libXML v2 development package, build the osgTDS plugin by starting in the osgTDS directory and executing the following commands:

```
cd src/TDSLoader
make
```

To install the resulting binary, issue this command as root:

```
make install
```

This installs the osgTDS plugin to the default OSG plugin directory.

Building and Installing for Windows Platforms

In `<path_to_osgTDS>\src`, you will find subfolders for different versions of the Microsoft Visual Studio compiler that contain workspace or solution files for the osgTDS project. Open the appropriate workspace or solution file for your compiler. Under the Build menu, select Batch Build and build everything.

Next, install the osgTDS plugin to a location where OSG can find it at runtime. After building, the osgTDS plugin is located in `<path_to_osgTDS>\bin\win32`. You can add this directory to your PATH or manually copy the osgTDS plugin to the existing osgPlugins folder.

2.1.2 The .TDS File Format

The TDS file format specifies the source terrain database as well as the targets that will potentially deform it. TDS uses the XML format, and the targets and terrain database exist as XML objects within the file. You can create or edit a TDS file with any text editor.

A Simple .TDS File

As an example, consider the following simple .TDS file:

Listing 1
A Simple .TDS File
This file configures osgTDS to deform terrain according to the elevations of two targets, house.osg and watertower.osg.

```
<?xml version="1.0" encoding="UTF-8"?>
<TDS>

<TargetDatabase>

    <Target
        FileName="house.osg"
    />
    <Target
        FileName="watertower.osg"
    />

    <Base>
        terrain0.osg
    </Base>

</TargetDatabase>

<PreservedFeatures>
</PreservedFeatures>

<Terrain
    FileNamePattern="terrain[0-9].osg"
/>

</TDS>
```

The file specifies a target database consisting of two polygonal targets defined by house.osg and watertower.osg, as well as a terrain file name pattern "terrain[0-9].osg". The file name pattern uses regular expression syntax to specify the set of terrain database files for osgTDS to deform. The target database specifies targets with known accurate locations, which osgTDS uses to create the correction function.

Note that the root element of the .TDS XML file is <TDS>. The <TDS> element has three child elements, <TargetDatabase>, <PreservedFeatures>, and <Terrain>, in any order. osgTDS requires the <TargetDatabase> and <Terrain> elements, but <PreservedFeatures> is optional.

The Target Database

The target database enumerates entities that osgTDS uses to create the correction function. Specify the target database with the <TargetDatabase> element. <TargetDatabase> must be a child of the <TDS> root element.

Specify the individual targets within the database as child nodes of the <TargetDatabase> element. <TargetDatabase> has two types of child elements: <Target> and <Base>.

The Target Element

Individual targets are defined by their file names. For example:

```
<Target FileName="bunker.shp" />
```

osgTDS also allows specification of invisible targets. This is useful in deformation of terrain by a single point primitive. The Target element supports an optional Boolean attribute to control its display. For example:

```
<Target FileName="point.osg" Display="false" />
```

This target is defined by the file point.osg. osgTDS uses the point.osg data to deform the terrain, but the geometry in point.osg isn't rendered.

The Base Element

This element specifies the terrain tiles under the targets. osgTDS uses these tiles to compute the elevation differences between the targets and the terrain. The Base element may contain multiple files. For example:

```
<Base>
    Terrain0.osg
    terrain1.osg
</Base>
```

In this case, osgTDS loads both terrain0.osg and terrain1.osg, and queries their elevations to determine the correction value at each target.

The results are undefined if the .TDS file specifies a target, but the Base element doesn't specify a terrain model corresponding to that target location. osgTDS will display the following warning in this case:

```
Terrain Deformation Software:  ERROR in Target Database Base
    definition:  at least one target is not completely over terrain.
```

Preserved Features

The preserved features element allows osgTDS to preserve features that aren't explicitly tagged in the OpenFLight terrain database. Typically, this element is used to make ridgeline geometry explicit. The preserved features element is optional; it may be empty or absent altogether.

The following example demonstrates how to use the preserved features element to preserve the geometry within the ridge.osg:

```
<PreservedFeatures>
    <Ridgeline FileName="ridge.osg" />
</ PreservedFeatures>
```

In this example, the preserved features element contains a ridgeline element. Multiple ridgeline elements may be present. The ridgeline sub-element is the only element currently supported for the preserved features element.

For more information on preserving ridgelines and other features, see section **2.2.3 Feature Preservation.**

The Terrain Database

The terrain database element specifies a regular expression pattern that defines the terrain file names for osgTDS to deform. The osgTDS plugin tests the name of every file loaded via the osgDB for a match against this pattern. If the file name matches the pattern, osgTDS loads the file and potentially deforms it.

A typical OpenFlight terrain database is an array of individual OpenFlight files that define a region of terrain. Their file names typically encode the location of the file geometry within the array. For example, "flight_08_11.flt" might be the OpenFlight file in row 8 and column 11 of the database array.

Specify the terrain database with the <Terrain> element. <Terrain> must be a child of the <TDS> root element. The <Terrain> element has a single attribute that specifies a file name pattern. For example:

```
<Terrain
    FileNamePattern="terrain_[0-9]_[0-9].flt"
/>
```

This element specifies a terrain database composed of files with names that match the pattern "flight_[0-9]_[0-9].flt". A file named "terrain_2_7.flt" would match this pattern. A file named "house.flt" would not match this pattern.

2.1.3 A Simple osgTDS Example

The example described in this section requires the osgviewer application. If you have an OpenSceneGraph source code tree, you can build osgviewer from source.

If you don't have the source code tree, you'll need to download and install the OSG binary distribution. Obtain OSG v1.2 binaries for your platform from the following URL:

```
http://www.openscenegraph.org/osgwiki/pmwiki.php/Downloads/Downloads
```

After installing the OSG binaries and osgviewer on your system, run the osgTDS example by issuing the following command:

```
cd <path_to_osgTDS>\Test\Case1
osgviewer example.tds terrain0.osg
```

Type the directory where you installed osgTDS in place of `<path_to_osgTDS>.`

The osgvewer application loads and displays files. Its command line arguments specify the files to load. In this example, osgviewer loads the files example.tds and terrain0.osg, in that order, using the **osgDB::readNodeFiles()** entry point.

osgviewer loads example.tds first. example.tds specifies a single target (house.osg), as well as the terrain database pattern. The osgTDS plugin parses this file, loads the target, stores the file name pattern, builds the correction space mesh, and returns the loaded target to the osgDB.

When osgviewer loads the second file, the osgTDS plugin tests its file name against the stored terrain database pattern. In this example, the file name matches the pattern, so the osgTDS plugin loads the terrain data, corrects it to fit the target database, and returns the processed terrain.

When the osgDB has finished using the osgTDS plugin to load all requested data, the osgDB returns the scene graph to the osgviewer application. Subsequently, osgviewer displays the corrected terrain database along with the target as Figure 2-2 shows.

For comparison (as Figure 2-5 shows), issue the following command to see what this scene would look like without TDS processing to correct the elevation mismatch:

```
osgviewer house.osg terrain0.osg
```

This command causes osgviewer to load two files. osgTDS doesn't process the terrain, because osgviewer doesn't load a .TDS file. Subsequently, OSG never loads and configures the osgTDS plugin. Instead, OSG loads both the house.osg model and the terrain0.osg terrain database with the standard OSG plugin for loading OSG files. As a result, the terrain is unprocessed by osgTDS, and the house appears to float above the terrain.

Figure 2-2
osgTDS Example Screenshot
This image shows a simple osgTDS example created using the osgviewer application. The house in the target database has a higher elevation than the local terrain, and osgTDS has processed the terrain to fit the target.

Figure 2-3
Example Without TDS Processing
This is the same scene as displayed in Figure 2-2 without TDS processing. The house floats above the terrain, because the elevation mismatch is uncorrected.

2.2 osgTDS Notes and Features

The osgTDS implementation contains explicit support for several commonly encountered features of visual simulation databases. The following text describes these features and how osgTDS supports them.

2.2.1 Paged and Tiled Databases

osgTDS is optimized to support OpenFlight databases arranged as an array of terrain tiles that are paged in at runtime.

osgTDS defines the correction space mesh independent of the terrain database extent. As each tile is paged in at runtime, osgTDS tests it for intersection with the CSM, and processes the terrain tiles only if they intersect the CSM.

osgTDS can support targets located near tile boundaries, because the correction function is continuous. This allows Gaussian hill or valley corrections to seamlessly straddle tile boundaries.

2.2.2 Variable Level of Detail Support

Many OpenFlight terrain databases contain multiple levels of detail (LOD) for each terrain tile. osgTDS corrects all terrain LODs to avoid popping effects that would otherwise occur if adjacent LOD geometry groups contained abrupt changes. When correcting lower LOD geometry, osgTDS scales the ε_T triangle fit tolerance linearly as a function of the LOD's minimum distance. Greater LOD distances increase ε_T, which relaxes the fit tolerance. As a result, osgTDS corrects distant LOD geometry with fewer triangles than nearby LOD geometry. This ensures that the most accurate fit is available in the most critical viewing situations, and doesn't impede the system with excess geometry at lower levels of detail where it isn't needed.

2.2.3 Feature Preservation

Many OpenFlight terrain databases contain features such as lakes, rivers, roads, and ridgelines. When applying corrections to the terrain database, osgTDS must handle such features with care to preserve their natural quality.

Lakes, Rivers, and Roads

Lakes, rivers, and roads typically appear in OpenFlight databases as a collection of polygonal Face records with a Group record parent. For osgTDS to recognize these features, OpenFlight databases must tag the feature's top-most Group record with a Comment record containing one of the following keywords:

- `lake`
- `river`
- `road`

If osgTDS finds any of these keywords within the Comment record text, osgTDS flags geometry within the corresponding Group record hierarchy as a feature. osgTDS treats all features identically. osgTDS won't introduce new vertices within the feature perimeter and won't allow triangles to cross a feature. Furthermore, osgTDS will ensure that the Z values of feature perimeters remain unaltered.

Ridgelines

Ridgelines typically appear in OpenFlight databases as a carefully modeled set of vertices shared by several Face records. osgTDS allows preservation of ridgeline vertices using the preserved features element of the .TDS file. To preserve ridgeline vertices, create a separate 3D file containing the ridgeline geometry as a line primitive, and add this file as a ridgeline sub-element of the preserved features element.

You can also use this solution to update ridgeline appearance in the event that more accurate data becomes available.

2.2.4 OpenFlight and ESRI ShapeFile Support

osgTDS will operate on any input file format supported by OSG. However, users will typically supply terrain databases in OpenFlight format and specify targets with ESRI Shapefiles. This section discusses how osgTDS uses these file formats, and how you can optimize these data files to work best with osgTDS.

OpenFlight

Models that contain multiple LODs to represent a single piece of geometry should have all LOD records as children to a common parent Group record. This ensures that osgTDS will be able to select the highest LOD of the set for correction.

For proper feature preservation, the parent Group records of lake, river, and road features should have Comment records that store the lake, river, and road keywords.

ESRI Shapefile

There are no osgTDS-specific optimizations for ESRI Shapefiles.

2.2.5 Troubleshooting

When used properly, osgTDS should deform your terrain database so the terrain seamlessly matches the elevation of the specified targets. If you still see targets floating above or sinking into the terrain, ensure that your application is doing the following:

- Your application must load the .TDS file before loading the terrain database. Use a debugger with your application to ensure that the osgDB loads the .TDS file before your application tries to load the terrain database.
- If the osgDB returns an error when loading the .TDS file, check the standard output for error or warning messages. osgTDS uses OSG's Notify system to display errors, so set the OSG_NOTIFY_LEVEL environment variable to INFO and run your application again to ensure that OSG displays all relevant messages.
- Make sure that the target in question is specified correctly in the target database section of the .TDS file. Check for spelling errors in the target file name.
- Identify the name of the terrain database tile file in question and ensure its name matches the FileNamePattern attribute in the Terrain element of the .TDS file. If the file name doesn't match the pattern, typically the osgDB will load the file using another plugin. As a result, the terrain data won't be deformed by TDS.
- Your application must use the osgDB (usually with the osgDB entry point **readNodeFile()** or **readNodeFiles()**) to load terrain database files. Otherwise, the osgTDS plugin will not have the opportunity to deform the terrain data.

If you build the osgDB plugin from source, use a debugger to set breakpoints in key locations such as **TDSLoader::readNode()** or the CorrectionSpaceMesh constructor. See the next chapter, **3 osgTDS Code Internals,** for an overview of the osgTDS source code.

3 osgTDS Code Internals

Conceptually, the osgTDS source code consists of three major functional parts:

- Creating the correction space mesh to define the correction function.
- Processing (subdividing and correcting) the terrain.
- Using the OSG plugin interface and other support code.

The osgTDS plugin implements this functionality as a collection of C++ classes. The CorrectionSpaceMesh class creates and stores the CSM triangle mesh and implements the correction function. The TerrainProcessor class subdivides the terrain and adds the correction to each vertex. The TDSLoader class exposes this functionality as a plugin to the osgDB library. Other classes exist to provide additional support and utility functions.

Source code for the TDS plugin is located in `<path_to_osgTDS>/src/TDSLoader.` The following is a description of each source file:

- CorrectionSpaceMesh.cpp and CorrectionSpaceMesh.h

  These files define the correction space mesh support class, which creates and stores the CSM triangle mesh and implements the correction function.

- DelaunayWAR.cpp and DelaunayWAR.h

  These files provide a workaround for the OSG DelaunayTriangulator class in the osgUtil library. Specifically, these functions preserve vertices along straight edges that osgUtil::DelaunayTriangulator would otherwise discard.

- TDSFileValuator.cpp and TDSFileValuator.h

  The TDSFileValuator class parses a .TDS file and stores all configuration parameters within that file, including options, targets, and the terrain database file pattern.

- TDSLoader.cpp and TDSLoader.h

  These files support the two-stage TDS process: 1) Loading the TDS file and targets and creating the correction space mesh, and 2) loading and processing database terrain tiles.

- TerrainProcessor.cpp and TerrainProcessor.h

  These files implement the meat of the terrain processing algorithm, including functionality to subdivide the terrain and correct the elevation of terrain vertices within the CSM.

- TerrainProcessingUtils.cpp and TerrainProcessingUtils.h

  The classes and set of functions defined in these files provide support and utility functions to the overall TDS process. Functionality includes target bottom hull creation, database intersection, perimeter creation, and convex hull creation and management.

- TexCoordPreserver.cpp and TexCoordPreserver.h

  This class provides texture coordinates for new vertices introduced by TDS triangle subdivision.

- Regex.cpp and Regex.h

  This is a wrapper class for regular expression pattern matching. It's used by TDSLoader to identify files that match the terrain file name pattern specified in the .TDS file.

- Win32Regex.cpp and Win32Regex.h

This class implements regular expression pattern matching for Microsoft Windows operating systems, which lack regular expression support in their runtime libraries.

3.1 Algorithm Implementation

The following sections discuss how each part of the TDS algorithm is implemented in osgTDS.

3.1.1 The Correction Space Mesh

The CorrectionSpaceMesh class, defined in CorrectionSpaceMesh.cpp and CorrectionSpaceMesh .h, implements the correction space mesh in the TDS algorithm.

The class constructor, CorrectionSpaceMesh(), has three sections of code (all line numbers refer to the file CorrectionSpaceMesh.cpp):

1. The first section of code is a large for loop starting at line 31. This iterates over all targets in the target database and adds the target vertices to a list of vertices in the correction space mesh. If the target is a polygonal model, the vertices comprising its bottom hull are added to the list. Otherwise, the target must be a point or line feature, whose vertices are added without further computation.
2. The second section of code computes the largest square extent around the target vertices that osgTDS could deform. This code section starts at line 120, and ends with a **for** loop at line 189 that adds the four corner vertices of the extent to the list of correction space mesh vertices.
3. After identifying all vertices in the correction space mesh, the vertices can be triangulated. This happens from line 256 through line 274.

The correction space mesh defines the correction function. The CorrectionSpaceMesh class contains a member function, **getZCorrection(),** that identifies the triangle in the CSM containing the *x,y* parameter point. This function is defined at line 323.

After identifying the triangle containing the point in question, **getZCorrection()** obtains the correction value from the Triangle class. **Triangle::C()** implements the correction function at line 452. This code computes the weights at each vertex, multiplying each weight by the influence function, **f().**

3.1.2 The Gaussian Influence Function

As described in the previous section, the CorrectionSpaceMesh class implements the CSM, which defines the correction function. It contains a class Triangle, which is instantiated for every triangle in the CSM. Triangle defines a method **C()** that implements the correction function. The correction function depends on the influence function **f()** to compute the correction value.

The influence function **f()** can be any suitable function. For general purpose terrain deformation, a Gaussian filter is a suitable influence function. The Triangle class implements the Gaussian filter in its method **f(),** which is defined at CorrectionSpaceMesh.cpp line 419.

3.1.3 Creating New Vertices

The TDS algorithm tests each terrain triangle for fit to the correction function, and subdivides it if necessary to minimize error. The TerrainProcessor class contains the code that tests and subdivides triangles to fit the new terrain. All line numbers refer to the file TerrainProcessor.cpp.

The TerrainProcessor class defines a method **_processGeometry().** The following section, **3.2 Code Walkthrough,** looks at this function in detail. Starting at line 219, this code iterates over terrain triangles, and calls the **_subdivide()** method on each triangle. The **_subdivide()** method definition starts at line 127.

_subdivide() uses the **_testFit()** method to check a triangle for fit to the correction function. If the test passes, **_subdivide()** simply adds the triangle to a list of vertices that become part of the final corrected terrain. If the triangle does not fit, **_subdivide()** creates four triangles from the current triangle, and recursively calls itself.

The **_testFit()** method is defined starting at line 93. It checks each edge for fit to the correction function. Note the constant value **et,** defined as 0.5. This corresponds to ε_T, the acceptable triangle edge tolerance in meters, from the equation in section **1.1.2 Terrain ProcessingAT |Terrain Processing}.** If the correction value at the midpoint of the line computed using linear interpolation, is greater than **et,** then the fit is deemed inadequate and **_testFit()** returns false. If all three edges pass the fit test, **_testFit()** returns true.

3.1.4 Special Cases

If a terrain vertex falls within the bottom hull boundary of a polygonal target, the correction function at that point must be the same as for the bottom hull vertices. This situation is handled in TerrainProcessor.cpp at line 254, within the **_processGeometry()** method.

3.2 Code Walkthrough

To further illustrate how osgTDS implements the TDS algorithm, the following sections walk through the code for a simple use case.

3.2.1 The TDSLoader Class

When your application attempts to load a .TDS file, the osgDB library locates the osgTDS plugin and calls into its **readNode()** method. This method is defined in TDSLoader.cpp at line 52. A conditional on the file name extension defines the major structure of this method.

If the file name extension is .TDS, the code uses **_tdsfv,** an instantiation of the TDSFileValuator class, to parse the file. It then loads the target and base model files specified in the .TDS file. These models must be loaded to define the CSM, which happens at line 140. **readNode()** returns the loaded targets as a scene graph, which your application typically renders as part of its scene.

The **else** clause of the conditional loads and processes terrain database files. The osgTDS plugin tells the osgDB that it can load any file. This allows the osgTDS plugin to look at any file name the application attempts to load, and test it for a match against the terrain database file name pattern. If the file isn't part of the terrain database, its name doesn't match the pattern, and the osgTDS plugin rejects the file. This occurs at line 181. In that case, the osgDB tries to load the file with another plugin.

Line 160 tests the file name for a match with the terrain database file name pattern by calling the **isTDSTerrainFile()** method. This method uses the code in Regex.cpp and Regex.h to test the file name against the regular expression defining the terrain database.

If the file name matches the pattern, osgTDS loads the file by explicitly asking the osgDB library for the plugin that supports that file extension. The osgDB returns a scene graph for the loaded terrain file. osgTDS instantiates a TerrainProcessor object to deform the terrain according to the TDS algorithm. The section, **3.2.2 Processing the Terrain,** discusses this in detail.

3.2.2 Processing the Terrain

The TerrainProcessor class identifies features and non-features in the loaded terrain and processes the non-feature geometry in a way that preserves the feature geometry.

Feature Identification

The TerrainProcessor uses a custom OSG NodeVisitor to search the loaded terrain database for features and non-features. (The osg::NodeVisitor class is OSG's mechanism for traversing a scene graph.) The custom NodeVisitor is the appropriately named FindFeatures class.

The feature identification process is based on how OSG loads OpenFlight files. As an oversimplification of the OpenFlight loading process, OSG essentially creates an osg::Node in the scene graph for each primary record in the OpenFlight file. If that primary record contains a comment record, OSG creates an osg::Node::DescriptionList from the comment record text. osg::Node::DescriptionList is simply a std::vector<std::string>. OSG attaches the DescriptionList to the Node corresponding to the primary record.

As FindFeatures traverses the scene graph, it searches each Node's DescriptionList for the keywords indicating river, lake, or road features. After traversal, FindFeatures contains two lists of geometry, one containing geometry identified as a feature, and the other containing non-feature geometry.

Geometry Processing

For each non-feature Geode returned by the FindFeatures visitor, TerrainProcessor calls the **TerrainProcessor::_processGeode()** method. In OSG, Geodes are leaf nodes that contain the drawable geometry to be rendered. **_processGeode()** calls **_processGeomtry()** for each of the Geometry objects stored in the Geode.

TerrainProcessor::_processGeometry() is the heart of osgTDS, and puts the TDS algorithm into action. The function performs the following operations:

- It obtains a list of coordinate vertices from the Geometry object being processed.
- It instantiates the TexCoordPreserver class, which is used later to assign texture coordinates to new vertices created during subdivision.
- It processes the features identified earlier by the FindFeatures visitor. It adds them to a list of constraints to be used in the Delaunay triangulation process, and computes the perimeter of each feature, which is used later to preserve the elevation of those vertices.
- It performs a Delaunay triangulation on the coordinates, constrained by the feature geometry. This ensures that the algorithm only needs to deal with triangles and not other primitive types, such as quadrilaterals or polygons.
- The **_subdivide()** method is called on each triangle. As described earlier, **_subdivide()** tests each triangle for fit to the correction function, and if necessary, subdivides the triangle into smaller triangles to obtain a better fit.
- After subdivision, **_processGeometry()** has a list of all vertices that potentially require correction. For each of these vertices, it applies the correction function, which adjusts the elevation to fit the terrain deformed by the correction function.
- **_processGeometry()** iterates over all of the bottom hulls to check for the special case of a terrain vertex lying within the base of a polygonal target. If it finds such a vertex, its elevation is adjusted to match the bottom of the target. If any bottom hull vertex is within the boundary of the set of vertices currently being processed, the vertex is added to the list of coordinate vertices. This produces a complete list of all vertices that will appear in the final corrected surface for this Geometry object.
- **_processGeometry()** performs a Delaunay triangulation on this set of vertices. This corrected triangular mesh fits the original terrain deformed by the correction function.
- If a vertex in the final mesh matches one of the feature periphery vertices, the code adjusts its elevation to match the elevation of the feature vertex.

- As a final step, **_processGeometry()** creates a new Geometry object with the final set of processed vertices. It also creates texture coordinates for each vertex using the TexCoordPreserver, and returns a pointer to this new Geometry object.

The **_processGeode()** method replaces the current Geometry object with the return value from **_processGeometry().** This replaces the original unmodified terrain geometry with the new terrain deformed by the correction function. After all Geometry objects are processed, **_processGeode()** uses the osgUtil library's SmoothingVisitor to generate normals for all vertices.

After the TerrainProcessor completely traverses the scene graph, the TDSLoader returns the root node of the scene graph to the calling application.

4 Issue Tracking

This section enumerates issues and resolutions concerning TDS and the osgTDS implementation.

4.1 Issue Code Description

TDS-XXX: Issue relates to the TDS algorithm.

OSG-XXX: Issue relates to OSG, or the osgTDS implementation of the TDS algorithm.

4.2 Issue Log

Issue Code	**Reporting Date**	**Description and Resolution**
OSG-001	Feb 8, 2007	osgTDS assumes target vertices are in absolute coordinates. If a target model contans a transformation, osgTDS will use the untransformed vertices, and the terrain correction will appear at the wrong location.

5 Bibliography

Latham, Roy, Donald Burns, "Dynamic Terrain Modification Using A Correction Algorithm." Image 2006 Conference Proceedings, July 2006.

Martz, Paul, *OpenSceneGraph Quick Start Guide*. lulu.com, 2007

OpenSceneGraph Web site, `http://www.openscenegraph.org`

osgTDS Web site, `http://www.andesengineering.com/Projects/TDS/`

Meshing Research Corner Web site, `http://www.andrew.cmu.edu/user/sowen/mesh.html`

Triangle Web site, `http://www.cs.cmu.edu/~quake/triangle.html`