VS Terrain

Overview

The VS Terrain is an alternate method to provide a terrain representation to the math models. There is a callback that you can install via the VS API. The Unreal Plug-in utilizes this callback to bind the solver with the Unreal terrain data. The callback can also be used to provide the solver your own data, or a built-in implementation can be utilized via VSTERRAIN files (Figure 1). These files can be generated by the VS Scene Builder, or the VS Terrain Utility.

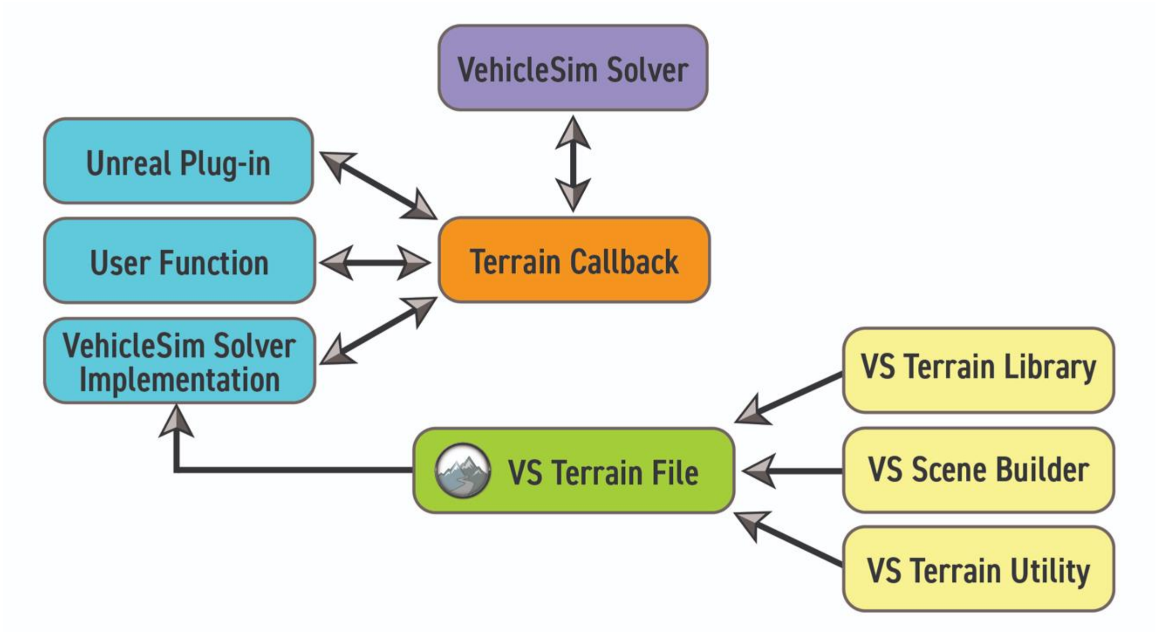


Figure 1. Overview of how VS Terrain is used in VS Solvers and Unreal.

VS Terrain Callback

You can install a terrain callback from your solver wrapper if you intend to use your own data and algorithms to supply the solver information about the drivable surface. The signature of the function needs to match `terrain_func_t` as defined in `vs_deftypes.h`.

```
typedef int (*terrain_func_t)(vs_real in_x, vs_real in_y, vs_real in_z,
                              vs_real *out_z, vs_real *out_dzdx, vs_real *out_dzdy,
                              vs_real *out_mu, vs_real *out_rr, void *userData);
```

The callback receives the xyz coordinates of the position to query. Your code should find the closest road surface directly above or below this point and return the z height along with the slope and surface properties (friction/mu and rolling resistance).

This function can then be installed via the VehicleSim API:

```
vs_road_install_terrain_function(terrain_func_t userFunction, void *userData)
```

The `userData` field will be passed back with each invocation of the installed callback and can be used as you see fit. Passing a handle to your terrain instance data would be a typical usage.

Note	Your callback should return non-zero if terrain was successfully found beneath the input coordinates (in_x, in_y, in_z). Zero can be returned to indicate no terrain was found at the given point, although you should still assign default values of your choosing to the outputs (out_*) for best results.

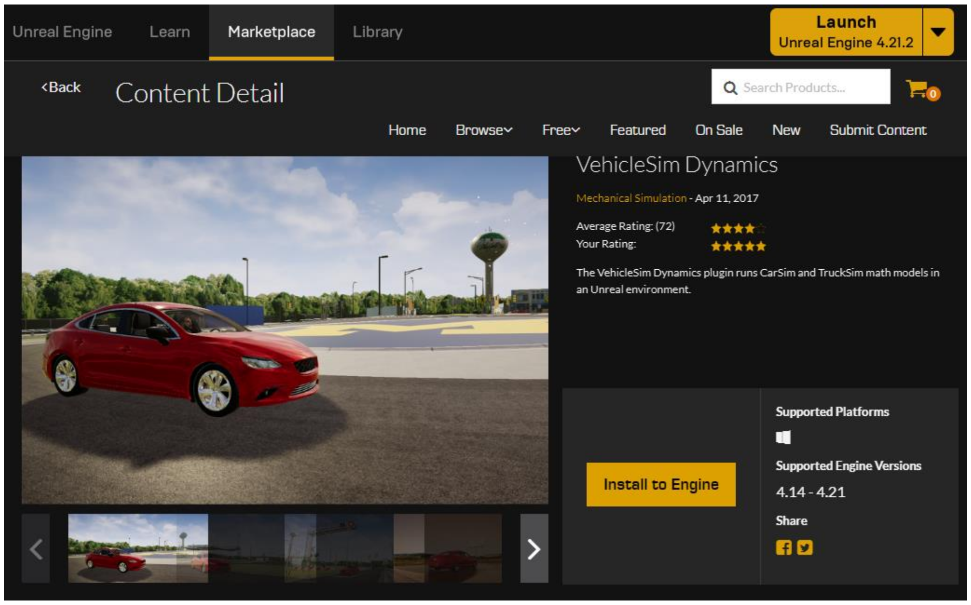
Unreal Engine Plug-in Usage of VS Terrain

The **VehicleSim Dynamics** plug-in for the Unreal Engine is provided for free by Mechanical Simulation on the Unreal Engine marketplace (Figure 2). This plug-in makes use of the VS Terrain callback to tie the terrain query capabilities of Unreal directly to the solver. More information on the plugin can be found in our guide **VehicleSim Dynamics plugin for Unreal**, within the VS Browser under **Help > Guides and Tutorials**.

Solver VS Terrain Implementation

The VS Solver contains a built-in implementation of the VS Terrain callback that can read in a VSTERRAIN file and will use this data to obtain surface information. At this time, you can generate VSTERRAIN files using the **VS Terrain Utility** provided by Mechanical Simulation. We may provide a header and DLL to directly create your own VS Terrain files from your own geometry in the future.

The first revision contained within the 2019.0 release uses a simple indexing scheme for the terrain geometry. This works fine for smaller terrain, but it can slow down the solver significantly as the size of the terrain grows.



VehicleSim Dynamics

Mechanical Simulation - Apr 11, 2017

Average Rating: (72)
Your Rating:

The VehicleSim Dynamics plugin runs CarSim and TruckSim math models in an Unreal environment.

Supported Platforms

Supported Engine Versions
4.14 - 4.21

Figure 2. The VehicleSim Dynamics plug-in on the Unreal Engine Marketplace.

The updates in 2019.1 introduced a more advanced indexing method that will scale more uniformly to large and/or sparse scenery. When you create a VS Terrain file, the geometry is first inspected and optimized to try to remove redundant point data. Mesh bundles are created based on their locality and density in certain regions. These bundles of geometry are placed into a spatial index, which can then be queried to narrow down candidates from a selected region and find the most appropriate terrain hit.

This query is used by the solver to find the height, friction, rolling resistance, and slope for a given location. When a query occurs, the solver will first test a vertical segment of several meters around the requested point. If it fails to find a valid terrain point, it will then try segments of several kilometers to locate any available terrain surface below, then above. If this still fails, it will just return the lower extent of the bounding box of the entire terrain geometry.

The VSTERRAIN file is provided to the solver via the VS_TERRAIN_FILE keyword, followed by the file name. For example, when using the **Scene: External Import** screen to import a VSSCENE file from the VS Scene Builder, if you open the parsfiles, you will see the command.

```
VS_TERRAIN_FILE Scene_Import\VS_Scene_Builder\test.vsterrain
```

Using VS Scene Builder to output VSTERRAIN files

The VS Scene Builder generates a VSTERRAIN file for the solver upon export. Each tile can contain its own VSTERRAIN file, so the VS Scene Builder will merge this data in the proper location for the export file.

VS Terrain Utility

The **VS Terrain Utility** is shown in Figure 3. It is provided so you can create drivable surfaces from your own shape files. The input format is FBX, a ubiquitous file format consumed and generated by many applications. It is broken into two sections, one for creating VS Terrain files and the other is for operations on those generated files.

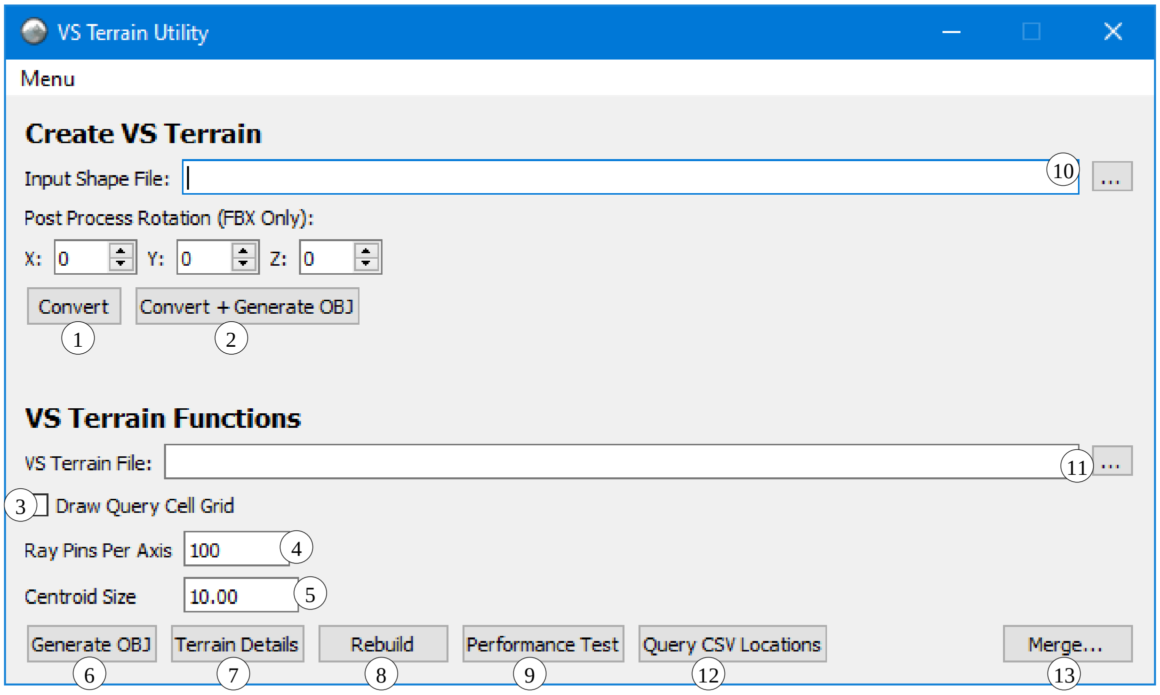


Figure 3. The VS Terrain Utility.

Create VS Terrain

Convert

If a valid FBX file is selected (10), pressing the Convert button (1) will generate a VS Terrain file. First a materials selection dialog will be presented that allow you to include/exclude geometry by material, as well as set the friction and rolling resistance.

Convert + Generate OBJ

This button (2) will run an FBX conversion and then automatically kick off generation of a debug OBJ file. This is equivalent to pressing Convert (1) followed by Generate OBJ (6).

VS Terrain Functions

Generate OBJ

The utility can also output an OBJ file (6) based on the selected VS Terrain File (11) that can be loaded in the **VS Visualizer** so you can inspect the surfaces that the math model is using. This OBJ file can contain indicators for the query cells (3). It can drop a number of test segments along the x and y axis to test for collision (4), which will be aligned along the surface normal at that location. You can also draw an indicator for the origin of the scene of a size you specify (5) in meters. See Figure 4 for an example of this output file loaded in VS Visualizer.

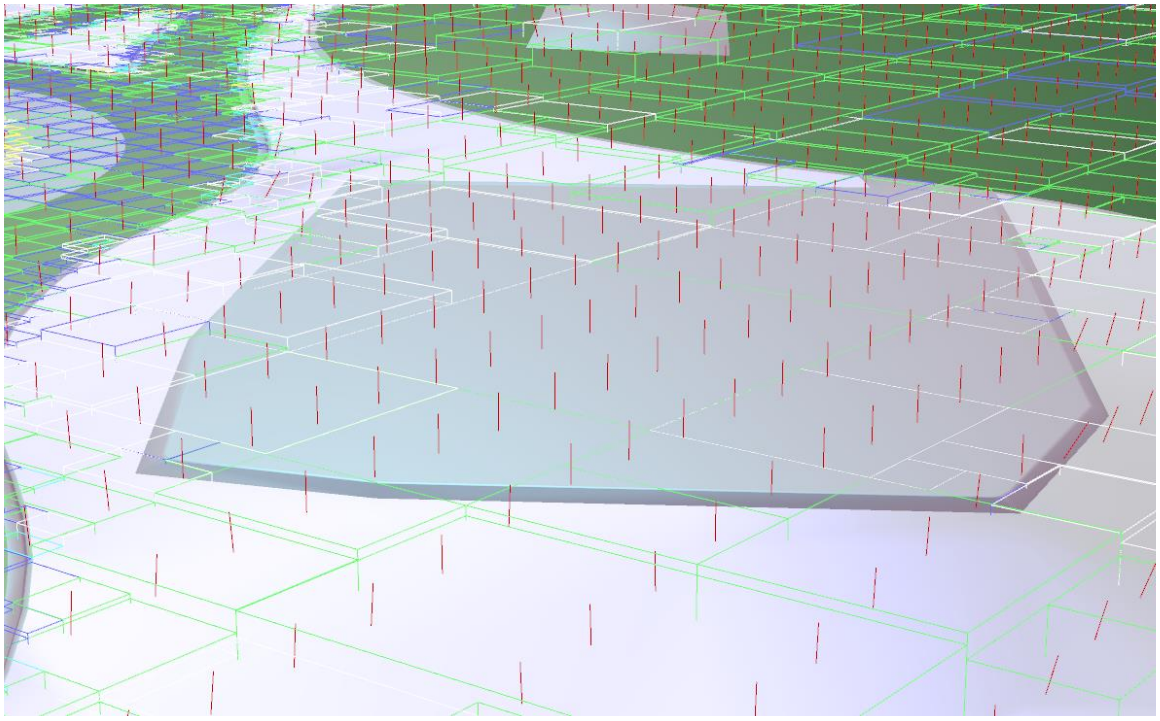


Figure 4. A scene overlaid with the Query Cell Grid data (boxes) and Ray Pins (red lines) from the VS Terrain Utility. See Table 5 for info on box colors.

Terrain Details

Pressing this button ⑦ will inspect the selected VS Terrain file ⑪ and display the basic details of the built file.

Rebuild

This takes an input VS Terrain file ⑪ and will rebuild it ⑧ with the current build settings.

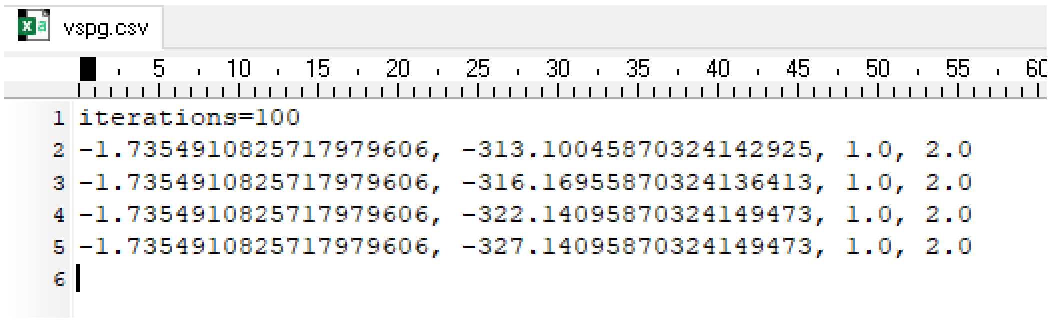
Performance Test

When this button ⑨ is pressed, the input VS Terrain file ⑪ is loaded, and millions of queries are performed while being timed. This can help determine the efficacy of build settings on a scene. Please note that this test does a series of uniform samples across the extents of the scene. If your drivable surface is surrounded by a lot of different-density geometry this can affect your average query time.

Query CSV Locations

This command ⑫ will prompt you to locate a comma-separated-value (CSV) file containing a series of segments to test against the input VS Terrain file ⑪. Each line in the CSV file is expected to define a segment to test or issue a command. The segment definition lines will provide `x`, `y`, `z`, and a `length` to test downward. This can allow for multiple terrain layers to be properly tested. The `z` and `length` parameters are optional. If length is not provided, a default distance above and below will be provided. If neither `z` or `length` is provided, a very large segment will be created above and below the given `x` and `y` coordinates.

The text file shown in Figure 5 will query the 4 points listed 100 times in a row and generate an output CSV file based on the input file name, appending `_out.csv` to the original name. This output will contain all queries along with data about the surface at that location if found. It will also generate timing information for the query at each location that can be used to find hot spots in a terrain.



```
iterations=100
-1.7354910825717979606, -313.10045870324142925, 1.0, 2.0
-1.7354910825717979606, -316.16955870324136413, 1.0, 2.0
-1.7354910825717979606, -322.14095870324149473, 1.0, 2.0
-1.7354910825717979606, -327.14095870324149473, 1.0, 2.0
```

Figure 5. An example input CSV file.

Merge…

The merge button (13) will prompt you to select multiple files as input, then select an output file that will be the merged terrain file.

Materials Dialog

The VS Terrain Utility has a single application menu that gives access to two dialogs: Materials and Build Settings (Figure 6).

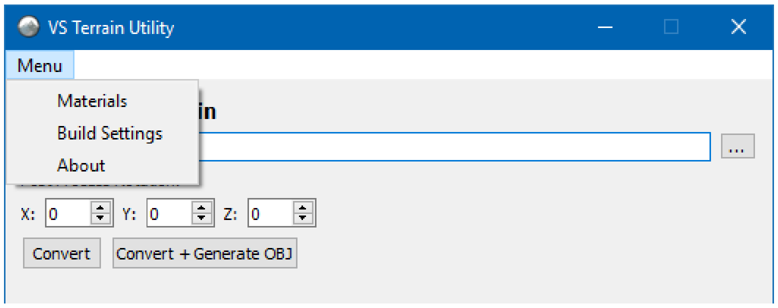
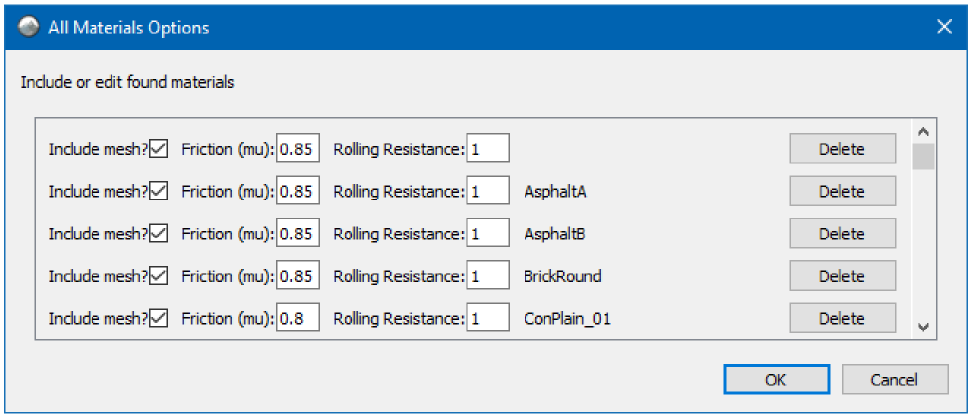


Figure 6. The application menu.

The materials dialog (Figure 7) is used to setup associations between material names and surface properties. The surface properties are friction, rolling resistance, and whether the geometry should be included in the built terrain file. This dialog can be accessed via the application menu and is also displayed when a build is started.



All Materials Options

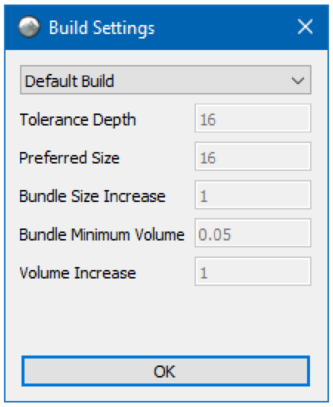
Include or edit found materials

Include mesh? ✓	Friction (mu): 0.85	Rolling Resistance: 1		Delete
Include mesh? ✓	Friction (mu): 0.85	Rolling Resistance: 1	AsphaltA	Delete
Include mesh? ✓	Friction (mu): 0.85	Rolling Resistance: 1	AsphaltB	Delete
Include mesh? ✓	Friction (mu): 0.85	Rolling Resistance: 1	BrickRound	Delete
Include mesh? ✓	Friction (mu): 0.8	Rolling Resistance: 1	ConPlain_01	Delete

Figure 7. The materials dialog.

Build Settings

The build settings have two preset values, as well as a custom mode. The default build settings are meant to work with the largest variety of scenes, producing a fast query time within the solver that builds quickly (Figure 8). The second preset is meant for higher density data. You may find that custom build settings work better for your data layout. See Table 4 (page 12) for more details on each parameter.



Build Settings

Default Build

Tolerance Depth	16
Preferred Size	16
Bundle Size Increase	1
Bundle Minimum Volume	0.05
Volume Increase	1

Figure 8. The build settings dialog.

Command-line Usage

The VS Terrain Utility can be used via the command-line, using the arguments listed in Table 1. Some examples are shown in Listing 1.

A JSON text files may be provided with build and material settings. Listing 2 shows an example settings file.

Please see the VS Scene Tile Creation documentation for an example of using the VS Terrain Utility to create a tile from your geometry.

Table 1. VS Terrain Utility command-line parameters

Command	**Description**
`-source [s]`	Specifies source file for a subsequent command.
`-target [t]`	Name of the terrain file to write using source data.
`-merge [m]`	Merge a list of vs terrain files into one file.
`-mu-multiply [mm]`	Multiply all terrain mu values by the multiplier mm
`-mu-override [mo]`	Override all terrain mu values with the value [mo]
`-rr-multiply [rrm]`	Multiply all terrain rolling resistance values by the multiplier rrm
`-rr-override [rro]`	Override all terrain rolling resistance values with the value [rro]
`-settings [s]`	Specifies the name of a settings file. The default file extension used is **.vstu**, but any valid JSON file is accepted. This file is a JSON file containing build and material settings for the build. See Listing 1 for an example.
`-queryfile [f]`	Instructs the VSTU to take the input file [f] and perform custom queries as described in the previous section. A source terrain file must also be specified with the `-s` parameter.
`-debug [d]`	Specifies an OBJ file with debug information to output for visual inspection.
`-pause [p]`	Pause for [p] seconds at the end of the build.
`-?, -h, --help`	Show help for command-line options.

Listing 1. VSTU command line examples.

```
--source shape.fbx --target track.vsterrain
--source shape.fbx --target track.vsterrain -settings urbanStyle.vstu
--source shape.fbx --target track.vsterrain --debug trackDebug.obj
--merge merged.vsterrain t1.vsterrain “t 2.vsterrain” t3.vsterrain ...
--debug trackDebug.obj --source track.vsterrain
```

Listing 2. Example .VSTU file for use with command-line processing.

```
{
  "VsTerrainUtility": {
    "Version": 1.0,
    "Depth Tolerance": 8,
    "Bundle Size Preferred": 8,
    "Bundle Increase": 0.5,
    "Minimum Bundle Volume": 0.125,
    "Bundle Volume Increase": 1.5,
    "Materials": [ {
      "Name": "Concrete",
      "mu": 0.8,
      "rr": 1.0,
      "include": 1
    }, {
      "Name": "Grass",
      "mu": 0.6,
      "rr": 1.0,
      "include": 1
    }, {
      "Name": "",
      "mu": 1.0,
      "rr": 1.0,
      "include": 1
    }]
  }
}
```

VS Terrain Library

The VS SDK includes DLL files for including the ability to create VS Terrain files directly from your application. A C header file is included, along with an example written in python that creates a simplified terrain. An example in C is included below in Listing 3 (page 13).

The functions available in the DLL are prefixed with `VST_API` in the header file and are documented in Table 2 and Table 3.

Table 2. VS Terrain Library basic functions

Function	**Description**
`vs_terrain_create`	Creates a terrain handle. Pass 0 to use the latest internal version, or a specific version if desired. The returned handle is used in all subsequent functions
`vs_terrain_destroy`	Cleans up the specified terrain handle.
`vs_terrain_add_triangle`	Adds the specified triangle to the terrain with the specified friction and rolling resistance. The triangles should be wound clockwise.
`vs_terrain_merge`	Copies the contents from a source terrain object to another. It is best to call optimize after this before doing a build.
`vs_terrain_optimize`	Calling optimize after all geometry has been added will try to optimize the mesh. You can provide a callback function of type **vs_terrain_optimize_callback** to be notified as progress is made.
`vs_terrain_build`	This builds the metadata necessary to query the structure.
`vs_terrain_build2_parameters`	Sets the build parameters to be subsequently used by vs_terrain_build2. See Table 4 for more detail.
`vs_terrain_build2_get_parameters`	Gets the build parameters to be subsequently used by vs_terrain_build2.
`vs_terrain_build2`	This build function allows for a progress callback and returns a results structure containing details about the build. Not compatible with terrain version 1.0.
`vs_terrain_save`	This saves the added terrain to the specified file. If the query metadata isn't built yet, it is built before writing the file.

Table 3. VS Terrain Library additional functions

Function	**Description**
`vs_terrain_clear`	Empties the specified terrain of all data.
`vs_terrain_get_version`	Returns the version of the terrain.
`vs_terrain_get_last_error`	If an error is thrown, returns the description.
`vs_terrain_get_num_triangles`	Returns the number of stored triangles.
`vs_terrain_get_num_points`	Returns the number of stored points.
`vs_terrain_get_triangle`	Returns the triangle data at the given index.
`vs_terrain_is_built`	Returns true if query metadata has been built.
`vs_terrain_get_extents_x`	Returns the min and max of the terrain along the x axis.
`vs_terrain_get_extents_y`	Returns the min and max of the terrain along the y axis.
`vs_terrain_get_extents_z`	Returns the min and max of the terrain along the z axis.
`vs_terrain_output_as_obj`	Outputs an OBJ file suitable for loading into the VS Visualizer to inspect your terrain. Setting **queryTestDropsPerAxis** to a positive value will determine how many “test queries” to run along the X and Y axis. These queries will be rendered as a red line that is normal to the face it collided with. Setting the **centroidSize** to a non-zero value will render a 3-axis triad at the origin of the scene, extending along the positive X, Y, and Z axis. If **drawQuery** is set to true, visualization of the query metadata will be included in the generated OBJ shape file.

When building the VS Terrain file, you can specify the parameters of the analysis to create a more optimal file for your application (Table 4). The VS Terrain header defines both a default group of settings, as well as a group optimized for high density data. The **VS Terrain Utility** includes a **Performance Test** that can help you test a terrain file.

Table 4. VS Terrain build parameters

Build Parameter	**Description**
`toleranceDepth`	To analyze a scene, the VS Terrain code will break the volume into smaller pieces. Each time this is done, the depth is incremented. The depth tolerance can be set to a value at which time the tolerances for accepting a volume, by triangle count or volume, will start to increase.
`preferredBundleSize`	When a volume is being examined, if the number of triangles occupying that space is at or below the specified value, the volume will be accepted. Otherwise, the algorithm continues to partition the space.
`toleranceDepthBundleIncrease`	For each step above the depth tolerance, the preferred bundle size will increase by this amount, rounded down to the nearest integer.
`minBundleVolume`	When a volume is being examined, if it occupies less than this minimum value it will be accepted, even if it contains more triangles than is being targeted.
`toleranceDepthVolumeIncrease`	For each step above the depth tolerance, the minimum bundle volume will be increased by this amount, expressed as a ratio of the minimum volume specified (i.e., 1.0 will increase the volume by 100% of the original value at each step).

When outputting the debug OBJ file, if the query structure is selected to be drawn, the query bins will be color coded by the number of triangles referenced within (Table 5).

Table 5. Bin box color reference

Query Bin Color	**Triangle Reference Count**
`White`	10 or less
`Green`	15 or less
`Blue`	20 or less
`Teal`	25 or less
`Yellow`	40 or less
`Purple`	60 or less
`Red`	More than 60

Any language that can bind to a DLL is able to use the VS Terrain Library. Included below is a basic C program creating a large, flat drivable surface. There is also a python example included with the VS SDK.

Listing 3. Source code for a C program using the VS Terrain Library.

```
#include “VsTerrain.h”

void optimize_callback(int progress, void *userData) { printf("\n%d\n", progress); }

int main()
{
  vs_terrain_point a = { -500.0, -500.0, 0.0 }, b = { 500.0, -500.0, 0.0 };
  vs_terrain_point c = { -500.0,  500.0, 0.0 }, d = { 500.0,  500.0, 0.0 };
  vs_terrain_mu_type mu = 0.85f;
  vs_terrain_rr_type rr = 1.0f;
  double weldThreshold = 0.001;

  vs_terrain_handle td = vs_terrain_create(0);
  vs_terrain_add_triangle(td, &a, &b, &c, mu, rr);
  vs_terrain_add_triangle(td, &c, &b, &d, mu, rr);
  vs_terrain_optimize(td, weldThreshold, optimize_callback, NULL);
  vs_terrain_build(td);
  vs_terrain_save(td, "example.vsterrain");
  vs_terrain_output_as_obj(td, "example.obj", 0, 0.0, false);
  vs_terrain_destroy(td);
  return 0;
}
```