HAPTIC PLUG-IN FOR UNITY

Institute	Digital Design Studio/Glasgow School of Art (http://www.gsa.ac.uk/research/research-centres/digital-design-studio/)
Project Supervisor	Dr. Matthieu Poyade
Development Team	Dr. Matthieu Poyade Michael Kargas Victor Portela
Contact Information	Dr. Matthieu Poyade (m.poyade@gsa.ac.uk) Digital Design Studio - Glasgow School of Art The Hub, Pacific Quay, G51 1EA, Glasgow, UK.

Table of Contents

Introduction 4
Haptic Plug-in for Unity - SOFTWARE DISCLAIMER 5
Installation 6
Examples 7
Manual of Use 9
Haptic Rendering 9
Modes of Interaction 10
Haptic Properties 11
Haptic Effects 12
References 13
Initialization and Clean Up Functions 13
bool InitHapticDevice() 13
bool HapticCleanUp() 13
Specify the Mode of Interaction 13
void SetMode(int mode) 13
int GetMode() 13
Haptic Object Face(s) - Touchable 13
void SetTouchableFace(IntPtr face) 13
Workspace Functions 14
IntPtr SetWorkspacePosition(IntPtr position) 14
IntPtr SetWorkspaceSize(IntPtr size) 14
void SetWorkspace(IntPtr position,IntPtr size) 14
void UpdateWorkspace(float CameraAngleOnY) 14
IntPtr GetWorkspacePosition() 14
IntPtr GetWorkspaceSize() 15
Render Haptics 15
void RenderHaptic() 15
Haptic Device and Proxy Values 15
IntPtr GetDevicePosition() 15
IntPtr GetProxyPosition() 15
IntPtr GetProxyRight() 15

IntPtr GetProxyDirection() 15
IntPtr GetProxyTorque() 16
IntPtr GetProxyOrientation() 16
Set Haptic Objects Mesh and Matrix Transform 17
void SetObjectTransform(int objectId,IntPtr name, IntPtr transform) 17
void SetObjectMesh(int objectId, IntPtr vertices, IntPtr triangles, int verticesNum, int trianglesNum) 17
Set Haptic Objects information 18
IntPtr GetHapticObjectName(int ObjId) 18
int GetHapticObjectCount() 18
int GetHapticObjectFaceCount(int ObjId) 18
Set Haptic Properties for Haptic Objects 18
void SetHapticProperty(int ObjId, IntPtr type , float value) 18
bool IsFixed(int ObjId) 19
Haptic Environmental Effects functions 20
bool SetEffect(IntPtr type,int effect_index, float gain, float magnitude, float duration,float frequency, IntPtr position, IntPtr direction) 20
bool StartEffect(int effect_index) 20
bool StopEffect(int effect_index) 20
Haptic Events Functions 21
void LaunchHapticEvent() 21
bool GetButton1State() 21
bool GetButton2State() 21
IntPtr GetTouchedObjectName() 21
int GetTouchedObjectId() 21
IntPtr GetManipulatedObjectName() 21
int GetManipulatedObjectId() 21
bool GetContact() 21
Puncture Mode Specific Functions 22
void SetMaximumPunctureLenght (float maxPenetration) 22
void SetPunctureLayers(int layerNb, IntPtr[] nameObjects , IntPtr layerDepth) 22
bool GetPunctureState() 22
float GetPenetrationRatio() 22
IntPtr GetFirstScpPt() 23
IntPtr GetPunctureDirection() 23

Introduction

This document presents the outcomes from a C++ development which enables implementing simple haptic interaction techniques using C# language in Unity Game Engine (http://unity3d.com).

The haptic interaction is ensured by the OpenHaptics Toolkit by Geomagic® (http://www.geomagic.com) formerly known as Sensable Technologies® and is only provided through Geomagic® haptic products.

The Development was carried out at the Digital Design Studio (DSS), part of the Glasgow School of Art, The Hub, Pacific Quay, G51 1EA.Glasgow, United Kingdom in June 2014 inspired by the development presented in the following PhD Thesis:

> *Poyade, M., (2013), Motor Skill Training using Virtual Reality and Haptic Interaction: a case study in industrial maintenance, PhD Thesis, University of Malaga, Malaga, Spain. (Available on Demand). Viva defended on July the 4th, 2013, University of Malaga, Malaga, Spain.*

Haptic Plug-in for Unity - SOFTWARE DISCLAIMER

The Haptic plug-in for Unity is provided FREE OF CHARGE by The Digital Design Studio (DDS), part of Glasgow School of Art, and has been tested only on a Geomagic® Touch, formerly known as Sensable Phantom Omni. The plug-in is not yet tested on any other Haptic Device. This software is supplied "AS IS" without any warranties and support.

DDS assumes no responsibility or liability for the use of the software, conveys no license or title under any patent, copyright, or mask work right to the product.

DDS reserves the right to make changes in the software without notification. DDS also makes no representation or warranty that such application will be suitable for the specified use without further testing or modification.

Any scientific contribution using the Haptic plug-in for Unity provided by the DDS, "MUST" be properly referenced to the DDS in corresponding publications under the following statement:

> *Haptic Plug-In for Unity (2014)- M. Poyade, M. Kargas, V. Portela, Digital Design Studio (DDS), Glasgow School of Art, Glasgow, United Kingdom.*

Additionally, any scientific contribution using the custom force effects provided by the Haptic plug-in for Unity for the simulation of power tools, "MUST" reference the following literature in corresponding publications:

> *Poyade, M., (2013), Motor Skill Training using Virtual Reality and Haptic Interaction: a case study in industrial maintenance, PhD Thesis, University of Malaga, Malaga, Spain. (Available on Demand). Viva defended on July the 4th, 2013, University of Malaga, Malaga, Spain.*

Any development using the Haptic plug-in for Unity provided by the DDS, for commercial purpose "MUST" be notified and agreed with the responsible persons at the DDS.

Installation

To proceed to the installation of the Haptic Plug-in for Unity in your project, please copy the following files into your Unity project Root folder.

From the OpenHaptics Toolkit files:

1. glu32.dll
2. glut32.dll
3. hd.dll
4. hl.dll
5. opengl32.dll
6. PhantomLib42.dll

You will find the Haptic Plug-in dll file "ASimpleHapticPlugin.dll" in the folder named "Plugins" which can be found in the Assets folder of the Unity example project.

Examples

The scenes corresponding to the examples mentioned below can be found in the Assets folder of the Haptic Plug-in for Unity Project in the folder named "Scene".

List of basic examples provided with the Haptic Plug-in for Unity:

1. Simple_Contact

A simple scene which allows touching different geometries and feeling the haptic characteristics assigned to each shape.

2. Basic_Shape_Manipulation

A simple scene of object manipulation which allows touching different geometries, feeling the haptic characteristics assigned to each shape, but also handling some of the presented objects perceiving their mass. Environmental force effects enable simulating the gravity, friction and viscosity applied into an environment, and a spring effect relative to a point into the 3D world space.

3. Basic_Shape_Manipulation_with_Physics

Same as the previous example, but the physics library of Unity is called to simulate some basic interactions between any object being manipulated and surroundings.

4. Custom_Force_Effect

A scene which uses some of the force effects designed to simulate the haptic sensations perceived during the manipulation of portable power tools, for instance right and straight angle portable polishing and grinding tools.

Force models for the simulation of such power tools have been presented and validated in the following literature:

Cuevas-Rodriguez, M., Poyade, M., Reyes-Lecuona, A., & Molina-Tanco, L. (2013). A VRPN server for haptic devices using OpenHaptics 3.0. In New Trends in Interaction, Virtual Reality and Modelling (pp. 73-82). Springer London.

Poyade, M., Molina-Tanco, L., Reyes Lecuona, A., Langley, A., Frutos, E., D cruz, M., Flores, S., (2012), Validation of a haptic virtual reality simulation in the context of industrial maintenance, Joint VR Conference of EuroVR and EGVE, 2012, Madrid, Spain. (Available on Demand).

Poyade, M., (2013), Motor Skill Training using Virtual Reality and Haptic Interaction: a case study in industrial maintenance, PhD Thesis, University of Malaga, Malaga, Spain. (Available on Demand).

This development was part of a doctoral work supported by the European 7th Framework project ManuVAR (Manual work support throughout system lifecycle by exploiting Virtual and Augmented Reality).

Poyade, M., (2013), Motor Skill Training using Virtual Reality and Haptic Interaction: a case study in industrial maintenance, PhD Thesis, University of Malaga, Malaga, Spain. (Available on Demand).

Krassi, B., D'Cruz, M., & Vink, P. (2010). ManuVAR: A framework for improving manual work through virtual an augmented reality. In Proceedings of the AHFE 3rd International Conference on Applied Human Factors and Ergonomics, AHFE, Miami, Florida, USA, 17–20 July, 2010.

5. Haptic_Injection (Proof of Concept Version)

A simple example of haptic puncture inspired by the implementation presented in:

Kellermann, K., Neugebauer, M., & Preim, B. (2011). A 6DOF Interaction Method for the Virtual Training of Minimally Invasive Access to the Spine. In Proceedings of CURAC, (pp. 143-148).

Anderson, P., Ma, M., & Poyade, M. (2014). A Haptic-Based Virtual Reality Head and Neck Model for Dental Education. In Virtual, Augmented Reality and Serious Games for Healthcare 1 (pp. 29-50). Springer Berlin Heidelberg.

List of advanced examples provided with the Haptic Plug-in for Unity:

1. Writing_Scene

A simple scene which allows drawing and writing on a black board using a wide variety of color.

2. Jenga_Scene

An entertaining application to play "Jenga" which uses the physics from Unity.

Manual of Use

Haptic Rendering

The implementation of the haptic interaction in Unity enables the haptic rendering of geometries considering the limitations of haptic rendering using the OpenGL Feedback Buffer. For more information, please see the OpenHaptics Programming Guide. It is required to specify the mesh of the touchable objects as well as the components of the corresponding transformation matrices to be set into the haptic frame using some of the functions detailed in the following sections.

To do so, it is required to respect the following sequence when developing your own haptic code in C#.

In C# Start function:

1. Launch of the Device.
2. Set up the Haptic Workspace Dimensions and Set Touchable Face(s) (Optional because default is set to front faces in the Haptic Plug-in)
3. Update Workspace (Set Workspace Orientation as a function of a Haptic Camera).
4. Set the Interaction Mode.
5. Set the Haptic Geometry (Mesh and Transformation Matrix).
6. Define the Environmental & Mechanical Effects (Eventually - These can be started and stop wherever in the C# code (e.g. On button events, On contact....)).
7. Launch Check-up for Haptic Events.

In C# Update function:

1. Set up & Update Workspace (if the workspace is supposed to change dynamically - otherwise, it is not useful).
2. Render the Haptic Frame (Client Thread in the Haptic Frame).
3. Get Haptic Device Proxy or Physical Tip Values.
4. Get Contact Information & Other Events (if needed).
5. Update Matrix transform of manipulated object (if needed).

In C# Disable Function:

1. Clean the Haptic Frame.

Environmental effects and other customized force effect can be implemented in the start function of your C# code and launched below the execution of some particular events, for instance, haptic device buttons pressed or haptic device touching a specific set of geometries....

Modes of Interaction

The haptic rendering of the Haptic Plug-In for Unity requires a mode of interaction to be defined prior to the rendering of the haptic frame.

Four modes of interaction are defined and supported by the Haptic Plug-In for Unity:

Mode 0. Simple Contact Mode (only haptic contact and basic environmental effects)

Mode 1. Object Manipulation Mode (enables the contact and manipulation of objects along with basic environmental force effects)

Mode 2. Custom Force Effects Mode (enables the contact and simulation of tangential and vibration force effects, and basic environmental force effects)

Mode 3. Puncture Mode (only contact, punctures and injections, and basic environmental force effects)

The haptic interaction supported by each mode is summarized in the table below:

Table 1. Functional summary of modes of interaction

Mode	name	Contact	Manip.	Env. Force Effects	Mechanical Force Effects	Puncture Injection
0	Simple Contact	Yes	No	Yes	No	No
1	Object Manipulation	Yes	Yes	Yes	No	No
2	Custom Force Effect	Yes	No	Yes	Yes	No
3	Puncture	Yes	No	Yes	No	Yes

Haptic Properties

Each touchable object is defined by a set of haptic properties which are:

Basic Haptic Properties defined in OpenHaptics:

1. **Stiffness** (float value between 0 and 1) - controls how hard surfaces feel - 0 represents compliant surface and 1 the hardest surface that can be rendered.
2. **Damping** (float value between 0 and 1) - reduces the springiness of the surface - 0 represents no damping and 1 the maximum level of damping.
3. **Static Friction** (float value between 0 and 1) - controls how hard it is to move onto the surface starting from a static position - 0 represents a frictionless surface and 1 the maximum amount of friction.
4. **Dynamic Friction** (float value between 0 and 1) - controls how hard it is to move onto a surface when motion has already been engaged - 0 represents a frictionless surface and 1 the maximum amount of friction.
5. **Pop Through** (float value between 0 and 1) - controls the amount of force to go through the geometry surface - 0 disables the pop through and 1 means that the maximum renderable force is required to go through.

Advanced Haptic Properties for Manipulation of Objects

1. **Mass** (float value between 0 and 1) - controls the weight of the object - 0 represents no mass and 1 the maximum renderable mass.
2. **Fixed Object** (boolean value 0 or 1) - set the object to fixed with 0 or manipulable with 1.

Advanced Haptic Properties for Custom Force Effects

1. **Tangential Stiffness** (float value between 0 and 1) - stiffness component of the tangential force caused by the contact of a rotating disc on a surface - controls how hard is a surface while simulating the functioning of a portable polishing or grinding tool - 0 represents no stiffness and 1 the maximum amount of hardness.
2. **Tangential Damping** (float value between 0 and 1) - controls the springiness of a surface - 0 represents no damping and 1 the maximum level of damping.

Advanced Haptic Properties for Puncture Effects

1. **Punctured Static Friction** (float value between 0 and 1) - controls how hard it is to move inside a punctured shape starting from a static position - 0 represents a frictionless tissue and 1 the maximum amount of friction.
2. **Punctured Dynamic Friction** (float value between 0 and 1) - controls how hard it is to move inside a punctured shape when motion has already been engaged - 0 represents a frictionless tissue and 1 the maximum amount of friction.

Haptic Effects

Using the Haptic Plug-in for Unity, it is possible to define several types of force effect which are:

Basic Environmental Force Effects:

1. **Constant Force Effect** - defines a constant force applied on the haptic device stylus - specified by a magnitude (float from 0 to 1), a direction (Vector3 - 0,-1,0 for gravity direction effect).
2. **Friction Force Effects** - defines a friction force applied to the haptic device in the free space - specified by a gain (float from 0 to 1) and a magnitude (float from 0 to 1).
3. **Spring Force Effect** - defines a spring force with regards to a specific position in the 3D world - specified a gain (float from 0 to 1), a magnitude (float from 0 to 1) and a position in the real world haptic workspace in millimetres (Vector3).
4. **Viscosity Force Effect** - defines a viscous force when moving in the free space, as if it was honey or liquid soap - specified by a gain (float from 0 to 1) and a magnitude (float from 0 to 1).

For all basic environmental effects, a duration of execution can be also specified (0 - always, >0 triggered to the indicated time).

Mechanical Force Effects:

1. **Motor Vibration Force Effect** - defines the vibration generated by the engine of a portable power tool (applied on the local x and y axis of the haptic device) - specified by a magnitude (float from 0 to 1) and a frequency (float) ***- A too high frequency can be harmful for the mechanics of the haptic device.***
2. **Contact Vibration Force Effect** - defines the vibration generated by a portable power tool when touching a geometry (applied on the normal axis of the geometry at the contact point) - specified by a magnitude (float from 0 to 1) and a frequency (float) - A too ***- A too high frequency can be harmful for the mechanics of the haptic device.***
3. **Tangential Force Effect** - defines the tangential force resulting from a rotating disc on the surface of an object - specified by a magnitude (float from 0 to 1) and function of tangential stiffness and tangential damping haptic properties (see previous section), and a direction (Vector3) to indicate if the simulated power tool is a right angle power tool (0,1,0) or a straight angle power tool (any other value) ***- When simulating tangential force, some sudden changes in force direction are to be expected, as function of the inclination of the simulated rotating disc onto objects surface - Handling firmly the haptic device stylus.***

References

Initialization and Clean Up Functions

bool InitHapticDevice()

Initialize the haptic context and launch the first haptic device found.

bool HapticCleanUp()

Clean the haptic context.

Specify the Mode of Interaction

void SetMode(int mode)

Set the mode of interaction:

0 - Simple Contact

1 - Object Manipulation

2 - Custom Force Effects

3 - Puncture Mode

int GetMode()

Return the current mode of interaction.

Haptic Object Face(s) - Touchable

void SetTouchableFace(IntPtr face)

Corresponding function in C++:

void SetTouchableFace(char* face);

Arguments required: IntPtr converted from a string(accepted values: front, back, front_and_back otherwise value is set by default to front).

Workspace Functions

IntPtr SetWorkspacePosition(IntPtr position)

Set the Workspace position.

Corresponding function in C++:

void SetWorkspacePosition(float position)*

Arguments required: float[3] type for x,y,z position in the world space.

IntPtr SetWorkspaceSize(IntPtr size)

Set the Workspace size.

Corresponding function in C++:

void SetWorkspaceSize(float size)*

Arguments required: float[3] type for scale on x,y,z axis.

void SetWorkspace(IntPtr position,IntPtr size)

Set the Workspace Position & Size.

Corresponding function in C++:

void SetWorkspace (float position, float* size)*

Arguments required: two float[3] type for position and scale.

void UpdateWorkspace(float CameraAngleOnY)

Update the workspace orientation as a function of the orientation on the Y axis of a "haptic camera". For more information, see example scripts.

IntPtr GetWorkspacePosition()

Return the position of the workspace - need to be converted into Vector3 format.

IntPtr GetWorkspaceSize()

Return the size of the workspace - need to be converted into a Float3 or Vector3 format.

Render Haptics

void RenderHaptic() *- To be used in the unity loop*

Draw the Haptic Scene according to the geometry information sent previously. For more information, see example scripts.

Haptic Device and Proxy Values

IntPtr GetDevicePosition()

Return the Position of the device - need to be converted into a Vector3 format.

IntPtr GetProxyPosition()

Return the position of the proxy which is the position of the graphical representation of the tip of the device onto the surface of a touched geometry.

When moving into the free space (no collision with shape), the proxy position and device position are the same. However, when touching a shape, proxy position and device position differ.

Need to be converted into a Vector3 format.

IntPtr GetProxyRight()

Return the right (X) orientation vector of the haptic device - need to be converted into a Vector3 format.

IntPtr GetProxyDirection()

Return the Direction (Z) orientation vector of the haptic device - need to be converted into a Vector3 format.

IntPtr GetProxyTorque()

Return the Torque (Y) orientation vector of the haptic device - need to be converted into a Vector3 format.

IntPtr GetProxyOrientation()

Return the Quaternion orientation vector of the haptic device - need to be converted into a Quaternion format.

Set Haptic Objects Mesh and Matrix Transform

void SetObjectTransform(int objectId,IntPtr name, IntPtr transform)

Set the transformation matrix of a touchable and manipulable object.

For more information, see script "GenericFunctionsClass.cs".

Corresponding function in C++:

void SetObjectTransform(int objectId, char name, double *transform)*

Arguments required: integer for the id of the object, a Char type as the name of the object, and a double[16] format matrix transform.

void SetObjectMesh(int objectId, IntPtr vertices, IntPtr triangles, int verticesNum, int trianglesNum)

Set the Mesh of a touchable and manipulable object.

For more information, see script "GenericFunctionsClass.cs".

Corresponding function in C++:

*void SetObjectMesh(int objectId, float *vertices, int *triangles, int verticesNum, int trianglesNum)*

Arguments required: integer for the id of the object, an float Array of vertices, a int Array of triangles, two integers which state for the length of both array.

Note that the Meshes of "GameObjects" accepted by this development consist of triangle.

Set Haptic Objects information

IntPtr GetHapticObjectName(int ObjId)

Return the name of the haptic object identify with the Id set as argument. - need to be converted into string format.

int GetHapticObjectCount()

Return the number of haptic objects.

int GetHapticObjectFaceCount(int ObjId)

Return the number of faces of a haptic object.

Set Haptic Properties for Haptic Objects

void SetHapticProperty(int ObjId, IntPtr type , float value)

Set a type of haptic property - generic function for any of the haptic properties mentioned below:

Basic properties: stiffness, damping, staticFriction, dynamicFriction

Advanced properties for manipulation of object: mass, fixed

advanced properties for custom forces effects: tangentialStiffness, tangentialDamping

Advanced properties to support the haptic puncture effect: popThrough, puncturedStaticFriction, puncturedDynamicFriction

Corresponding function in C++:

*void SetHapticProperty(int ObjId, char *type , float value)*

Arguments required: Integer for the Id of the object, a string converted to IntPtr, a float value between 0 to 1 (excepted for "fixed" is a boolean - so the fixed value must be converted to Int32 using System.Convert.ToInt32(bool)).

This function must be repeated for each type of haptic properties that needs to be defined - Otherwise, specific functions as those presented below can be used instead.

For more information, see script "GenericFunctionsClass.cs".

void SetStiffness(int ObjId, float stiffness)

void SetDamping(int ObjId, float damping)

void SetStaticFriction(int ObjId, float staticFriction)

void SetDynamicFriction(int ObjId, float dynamicFriction)

void SetTangentialStiffness(int ObjId, float tgStiffness)

void SetTangentialDamping(int ObjId, float tgDamping)

void SetPopThrough(int ObjId, float popThrough)

void SetPuncturedStaticFriction(int ObjId, float puncturedStaticFriction)

void SetPuncturedDynamicFriction(int ObjId, float puncturedDynamicFriction)

void SetMass(int ObjId,float mass)

void SetFixed(int ObjId,bool fix)

bool IsFixed(int ObjId)

Return the state of the Fixed value of a haptic object. true = fixed, false = manipulable.

Haptic Environmental Effects functions

bool SetEffect(IntPtr type,int effect_index, float gain, float magnitude, float duration,float frequency, IntPtr position, IntPtr direction)

Set type of effects - generic function to define effect. Possible effects presented below:

constant, spring, viscous, friction, vibrationMotor, vibrationContact, tangentialForce,

Corresponding function in C++:

*void SetEffect(char *type,int effect_index, float gain, float magnitude, float duration, float frequency, float position[3], float direction[3])*

Arguments required: A string for the type that must be converted to IntPtr, an integer for the index of the effects - note that, if there is several effects, number Id must be sequential, starting from 0, a float for the gain, a float for the magnitude, a float for the duration (if null then effect will not stop until the stopEffect is executed), a float for the frequency, a float[3] for the position, and finally a float[3] for the direction. Note that not all the arguments are used when defining an effect. For more information, report to previous sections.

For more information, see script "GenericFunctionsClass.cs".

bool StartEffect(int effect_index)

Start the identified effect - Effects set using the "SetEffect" function need to be launched using the "StartEffect" function.

bool StopEffect(int effect_index)

Stop the identified effect.

Haptic Events Functions

void LaunchHapticEvent()

Launch the Event Checker - Necessary if we pretend to get the contact and button information. For more information, see example scripts.

bool GetButton1State()

Return the state of button1.

bool GetButton2State()

Return the state of button2.

IntPtr GetTouchedObjectName()

Return the name of the touched object - need to be converted to string.

int GetTouchedObjectId()

Return the Id of the touched object.

IntPtr GetManipulatedObjectName()

Return the name of the manipulated object - need to be converted to string.

int GetManipulatedObjectId()

Return the Id of the manipulated object.

bool GetContact()

Return the state of contact - true: contact with geometry, false: no contact.

Puncture Mode Specific Functions

void SetMaximumPunctureLenght (float maxPenetration)

Set the Maximum depth of penetration of the needle/syringe into geometries.

void SetPunctureLayers(int layerNb, IntPtr[] nameObjects , IntPtr layerDepth)

Set the layers composed by the different shapes which are found in the Puncture stack. Arguments required for this function can be obtained with a RayCastAll in Unity. For more information, see script "HapticInjection.cs"

Corresponding function in C++:

*void SetPunctureLayers(int layerNb, char** name, float* layerDepth)*

Arguments required: An integer that indicates the number of hits returned by the raycasting, An array of IntPtr build from a collection of string (string[] in C#), and an array of float converted to IntPtr which states for all the hitting distances returned by the raycasting.

Note that the RayCastAll function does not guarantee to return data in a specific order - However, the plug-in sorts the array.

As well, the plug-in generates a penetration restriction when a geometry (the first found in the puncture stack) with pop-through component equals to null is found in the puncture stack. That penetration restriction compensates the fact that the Proxy Method along such constraint line is somehow inaccurate - most probably due to the difference between device position and proxy position because the constraint applies forces onto the device. So, the plug-in impedes the proxy to penetrate in underlying layers when their pop-through values is null.

bool GetPunctureState()

Return the state of the puncture - true: A geometry is being punctured, false: no puncture.

float GetPenetrationRatio()

Return the ratio between the current depth of penetration and the maximum depth of penetration allowed.

IntPtr GetFirstScpPt()

Return the position of the first puncture point - need to be converted to Vector3 format.

IntPtr GetPunctureDirection()

Return a vector that points in the direction of the puncture - need to be convert to Vector3 Format.