Houdini × Python

2019年9月6日　山部 道義

自己紹介

山部 道義

経歴：

　映像制作の分野でCGデザイナーとしてTV、遊技機映像など多数の映像制作に携わる。
　2009年よりSony Interactive Entertainment Worldwide Studios JAPAN Studioにてテクニカルアーティストとして、主にDCCツール向けの開発とパイプライン開発に従事。
　2016年より映像制作分野へ転身しHoudiniを用いた群衆シミュレーションやエフェクト制作に従事。
　2018年より株式会社ポリフォニー・デジタルにて、アーティストの作業を支えるパイプライン開発、ツール開発に従事。

CEDEC2018「プログラマーだってHoudini覚えたい！～プログラマーのプログラマーによるプログラマーのためのHoudiniトーーク」

POLYPHONY™ DIGITAL

今回の内容

- Houdini × Python 基礎知識
- Houdini × Python Cythonによる高速化について
- まとめ

Houdini × Python 基礎知識

HoudiniのPythonでできる主なこと

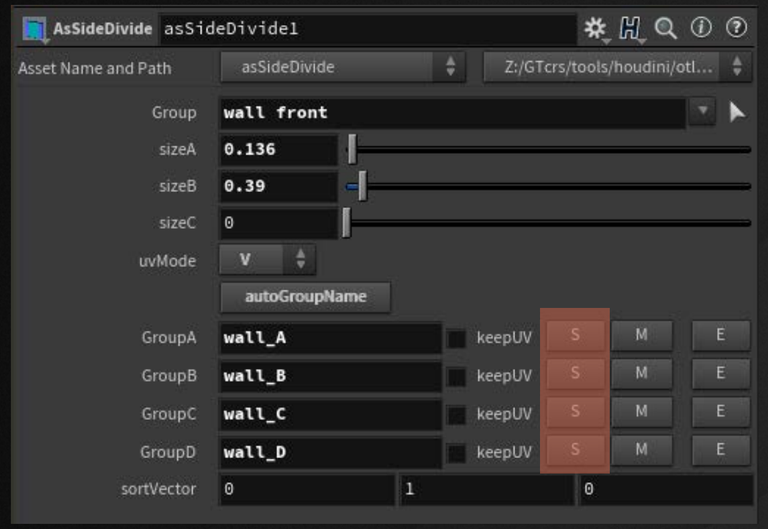
- スタンドアローンツールの作成
- バッチ処理
- カスタムノードの作成
- カスタムマニピュレータの作成
- パラメータコールバック
- パラメータエクスプレッション

など

まずは import hou から

- HoudiniのPythonは、まずはhouモジュールのインポートから始まります

AsSideDivide HDA ボタンコールバックの例




```python
def addSideDivie(groupName):
    node = hou.pwd()
    parent = node.parent()
    dstGroupName = node.parm(groupName).eval()

    newNode = parent.createNode("asSideDivide")
    newNode.parm("group").set(dstGroupName)
    newNode.setFirstInput(node)
    newNode.moveToGoodPosition()
    newNode.setDisplayFlag(True)
    newNode.setRenderFlag(True)
    newNode.setCurrent(True, clear_all_selected=True)
    newNode.hdaModule().autoGroupName(newNode)
```

POLYPHONY™ DIGITAL

Subdivide ボタンのコールバックの主な処理の流れ

- 現在のノードの階層情報を取得
- ノード作成
- ノード同士の接続
- ノードのパラメータを設定

```
def addSideDivie(groupName):
    node = hou.pwd()
    parent = node.parent()
    dstGroupName = node.parm(groupName).eval()

    newNode = parent.createNode("asSideDivide")
    newNode.parm("group").set(dstGroupName)
    newNode.setFirstInput(node)
    newNode.moveToGoodPosition()
    newNode.setDisplayFlag(True)
    newNode.setRenderFlag(True)
    newNode.setCurrent(True, clear_all_selected=True)
    newNode.hdaModule().autoGroupName(newNode)
```

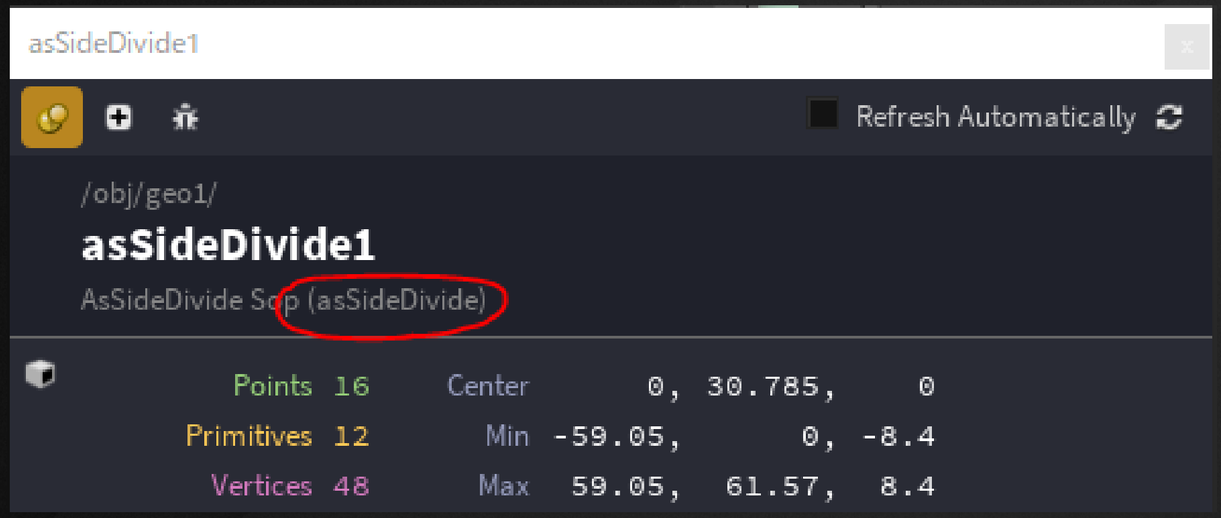
現在のノードに関する情報を取得

- スクリプトを起動した現在のノードを取得
  - node = hou.pwd()
- 現在のノードを含む親ノードを取得
  - parent = node.parent()
- 現在のノードにセットされているグループパラメータからグループ名を取得
  - dstGroupName = node.parm(groupName).eval()

```
def addSideDivie(groupName):
    node = hou.pwd()
    parent = node.parent()
    dstGroupName = node.parm(groupName).eval()

    newNode = parent.createNode("asSideDivide")
    newNode.parm("group").set(dstGroupName)
    newNode.setFirstInput(node)
    newNode.moveToGoodPosition()
    newNode.setDisplayFlag(True)
    newNode.setRenderFlag(True)
    newNode.setCurrent(True, clear_all_selected=True)
    newNode.hdaModule().autoGroupName(newNode)
```

ノード作成

- 新規ノードを親ノードの中に作る
  - newNode = parent.createNode("asSideDivide")




```python
def addSideDivie(groupName):
    node = hou.pwd()
    parent = node.parent()
    dstGroupName = node.parm(groupName).eval()

    newNode = parent.createNode("asSideDivide")
    newNode.parm("group").set(dstGroupName)
    newNode.setFirstInput(node)
    newNode.moveToGoodPosition()
    newNode.setDisplayFlag(True)
    newNode.setRenderFlag(True)
    newNode.setCurrent(True, clear_all_selected=True)
    newNode.hdaModule().autoGroupName(newNode)
```

POLYPHONY™ DIGITAL

ノード同士の接続

- 現在ノードと新規ノードの0番コネクタを接続
  - newNode.setFirstInput(node)
- 接続された新規ノードを、適切な位置へ配置
  - newNode.moveToGoodPosition()

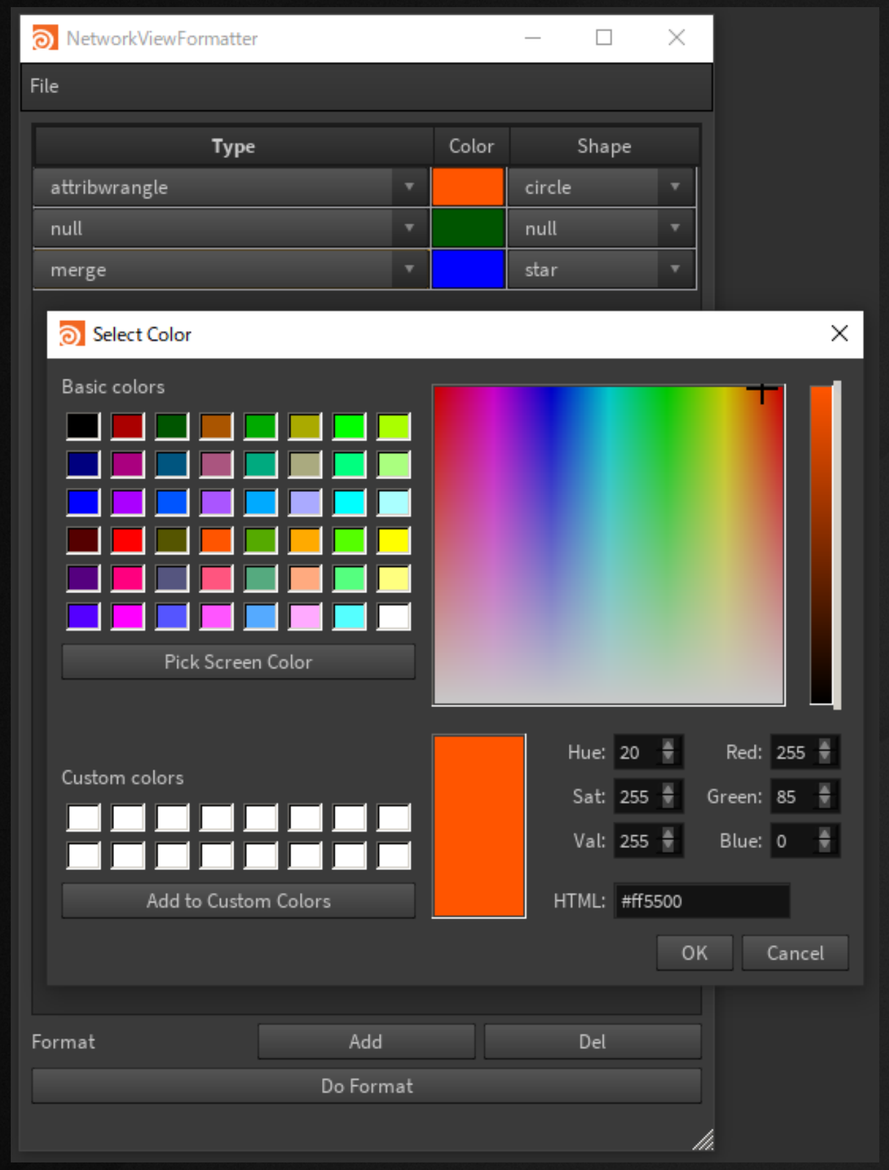
```
def addSideDivie(groupName):
    node = hou.pwd()
    parent = node.parent()
    dstGroupName = node.parm(groupName).eval()

    newNode = parent.createNode("asSideDivide")
    newNode.parm("group").set(dstGroupName)
    newNode.setFirstInput(node)
    newNode.moveToGoodPosition()
    newNode.setDisplayFlag(True)
    newNode.setRenderFlag(True)
    newNode.setCurrent(True, clear_all_selected=True)
    newNode.hdaModule().autoGroupName(newNode)
```

ノードの状態を設定

- グループパラメータをセットする
  - newNode.parm("group").set(dstGroupName)
- 新規ノードの表示・レンダーフラグをオンにする
  - newNode.setDisplayFlag(True)
  - newNode.setRenderFlag(True)
- 新規ノードを選択する
  - newNode.setCurrent(True, clear_all_selected=True)
- PythonModule内の関数を使ってGroupA-Dの内容をセットする
  - newNode.hdaModule().autoGroupName(newNode)

```
def addSideDivie(groupName):
    node = hou.pwd()
    parent = node.parent()
    dstGroupName = node.parm(groupName).eval()

    newNode = parent.createNode("asSideDivide")
    newNode.parm("group").set(dstGroupName)
    newNode.setFirstInput(node)
    newNode.moveToGoodPosition()
    newNode.setDisplayFlag(True)
    newNode.setRenderFlag(True)
    newNode.setCurrent(True, clear_all_selected=True)
    newNode.hdaModule().autoGroupName(newNode)
```

ノード外観整理ツール

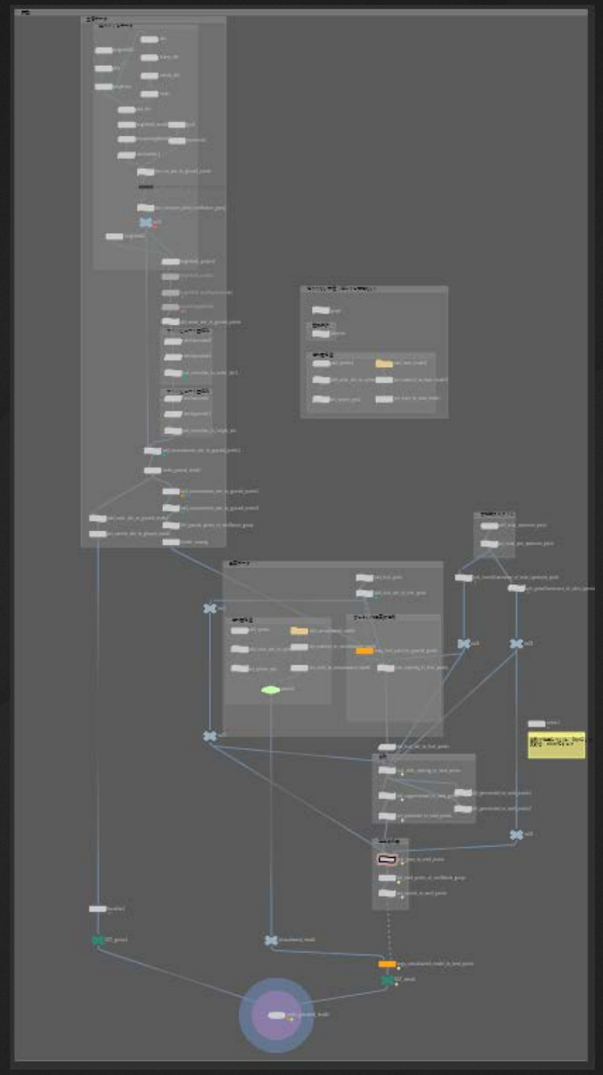


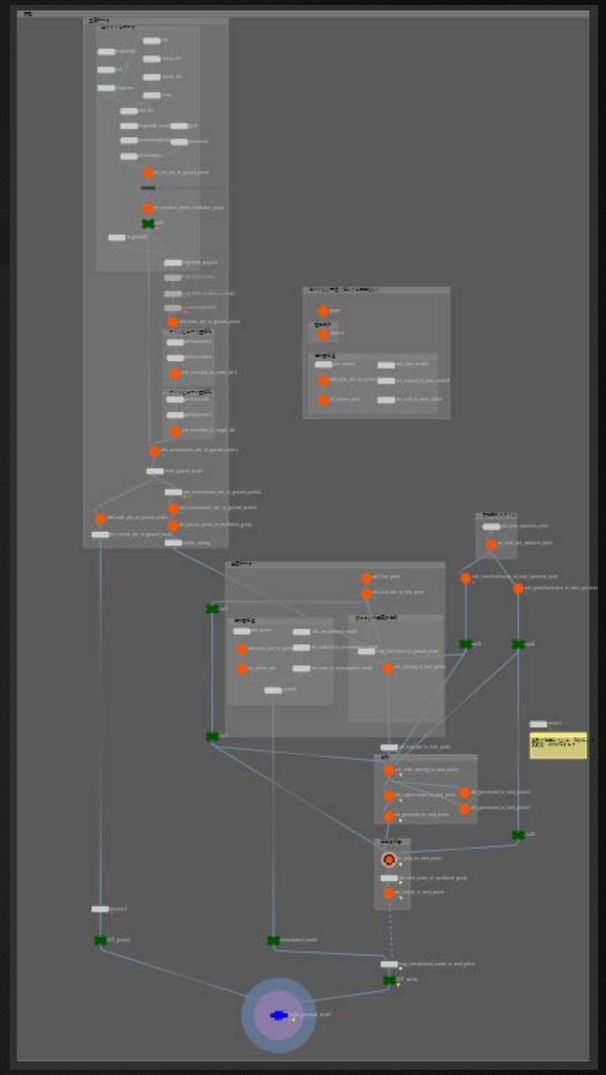
- ノードタイプ別にカラーとシェイプを事前設定
- ノードのカラーとシェイプを一括適用

POLYPHONY DIGITAL

ノード外観整理ツール

Type	Color	Shape
attribwrangle		circle
null		null
merge		star





ノード外観整理の主な処理のコード

```
def apply_format(self, *args, **kwargs):
    setting_dict = self.build_settings()
    target_type_names = setting_dict.keys()

    network_editor = hou.ui.paneTabOfType(hou.paneTabType.NetworkEditor)
    current_node = network_editor.pwd()
    children = current_node.children()

    for child in children:
        child_type_name = child.type().name()

        if child_type_name in target_type_names:
            setting = setting_dict[child_type_name]

            child_color = hou.Color(setting[0][0]/255.0,       You, a few s
                                    setting[0][1]/255.0,
                                    setting[0][2]/255.0)
            child.setColor(child_color)

            child_shape = setting[1]
            child.setUserData('nodeshape', child_shape)

        else:
            child.setUserData('nodeshape', "rect")
            child.setColor(hou.Color(0.8, 0.8, 0.8))
```

ノード外観整理の主な処理の流れ

- ネットワークエディタ内のノードリストを取得
- 各ノードに外観設定を適用

```python
def apply_format(self, *args, **kwargs):
    setting_dict = self.build_settings()
    target_type_names = setting_dict.keys()

    network_editor = hou.ui.paneTabOfType(hou.paneTabType.NetworkEditor)
    current_node = network_editor.pwd()
    children = current_node.children()

    for child in children:
        child_type_name = child.type().name()

        if child_type_name in target_type_names:
            setting = setting_dict[child_type_name]

            child_color = hou.Color(setting[0][0]/255.0,
                                    setting[0][1]/255.0,
                                    setting[0][2]/255.0)
            child.setColor(child_color)

            child_shape = setting[1]
            child.setUserData('nodeshape', child_shape)

        else:
            child.setUserData('nodeshape', "rect")
            child.setColor(hou.Color(0.8, 0.8, 0.8))
```

POLYPHONY™ DIGITAL

ネットワークエディタ内のノードリストを取得

```
def apply_format(self, *args, **kwargs):
    setting_dict = self.build_settings()
    target_type_names = setting_dict.keys()

    network_editor = hou.ui.paneTabOfType(hou.paneTabType.NetworkEditor)
    current_node = network_editor.pwd()
    children = current_node.children()

    for child in children:
        child_type_name = child.type().name()

        if child_type_name in target_type_names:
            setting = setting_dict[child_type_name]

            child_color = hou.Color(setting[0][0]/255.0,
                                    setting[0][1]/255.0,
                                    setting[0][2]/255.0)
            child.setColor(child_color)

            child_shape = setting[1]
            child.setUserData('nodeshape', child_shape)

        else:
            child.setUserData('nodeshape', "rect")
            child.setColor(hou.Color(0.8, 0.8, 0.8))
```

- ネットワークエディタを取得する
  - network_editor = hou.ui.paneTabOfType(hou.paneTabType.NetworkEditor)
- ネットワークエディタがフォーカスしているノードを取得する
  - current_node = network_editor.pwd()
- ネットワークエディタがフォーカスしているノードの子ノードリストを取得する
  - editor_nodes = current_node.children()

各ノードに外観設定を適用

- ノードのタイプ名を取得
  - node_type = editor_node.type()
  - type_name = node_type.name()
- ノードのカラー設定
  - editor_node.setColor(hou.color())
- ノードのシェイプ設定
  - child_node.setUserData('nodeshape', 'circle')

```
def apply_format(self, *args, **kwargs):
    setting_dict = self.build_settings()
    target_type_names = setting_dict.keys()

    network_editor = hou.ui.paneTabOfType(hou.paneTabType.NetworkEditor)
    current_node = network_editor.pwd()
    children = current_node.children()

    for child in children:
        child_type_name = child.type().name()

        if child_type_name in target_type_names:
            setting = setting_dict[child_type_name]

            child_color = hou.Color(setting[0][0]/255.0,
                                    setting[0][1]/255.0,
                                    setting[0][2]/255.0)
            child.setColor(child_color)

            child_shape = setting[1]
            child.setUserData('nodeshape', child_shape)

        else:
            child.setUserData('nodeshape', "rect")
            child.setColor(hou.Color(0.8, 0.8, 0.8))
```

POLYPHONY DIGITAL

サンプルからわかること

- ほとんどの重要な操作はhou.Nodeとhou.Parmの操作でできている
- 特別に難しい処理はしていない

Python実行環境

Python Source Editor
Shelf Tool
Python Node
Parameter Callback Script

など

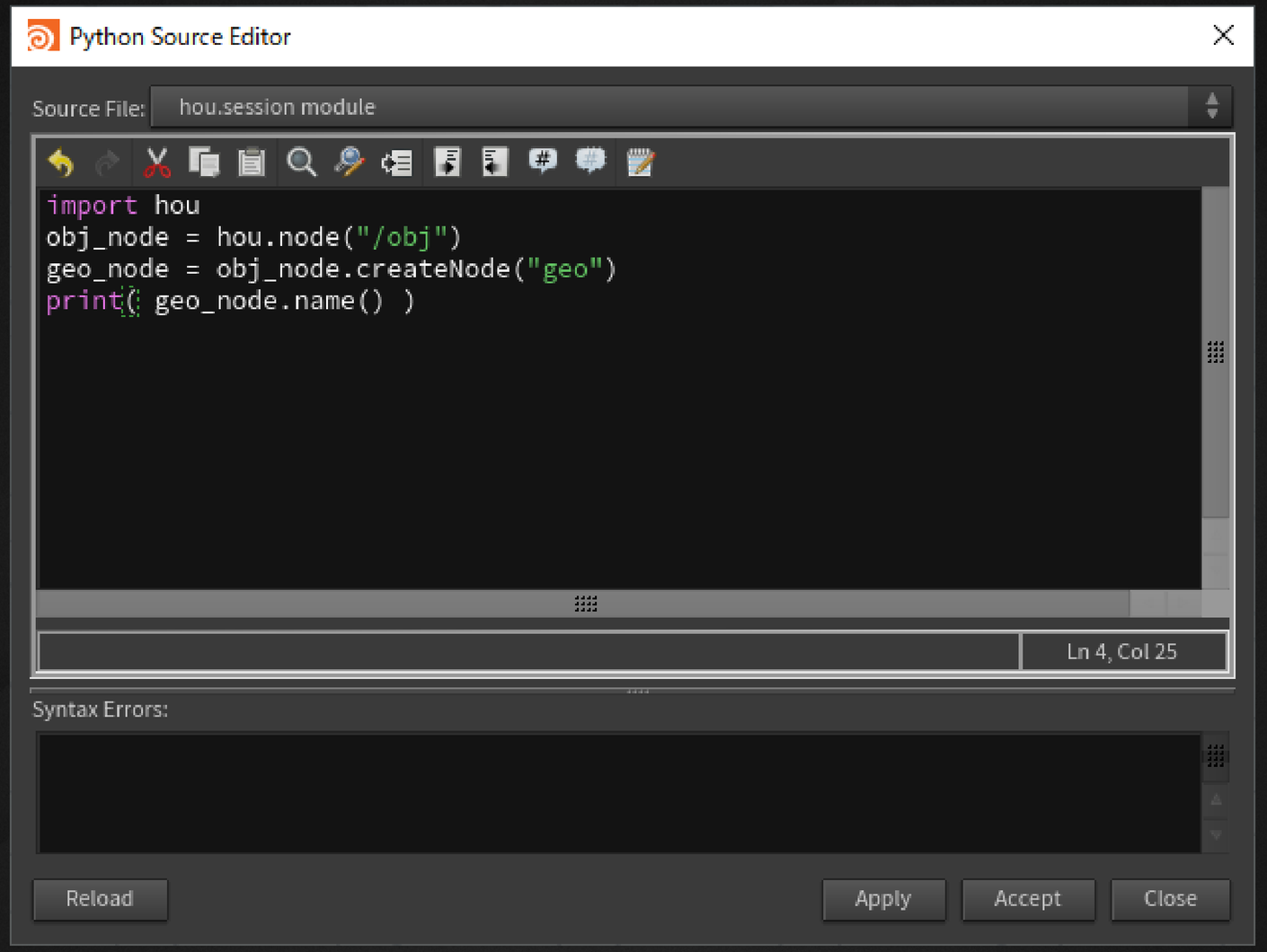
Python Source Editor

Shelf Tool

Python Node

Parameter Callback Script

Python Source Editor



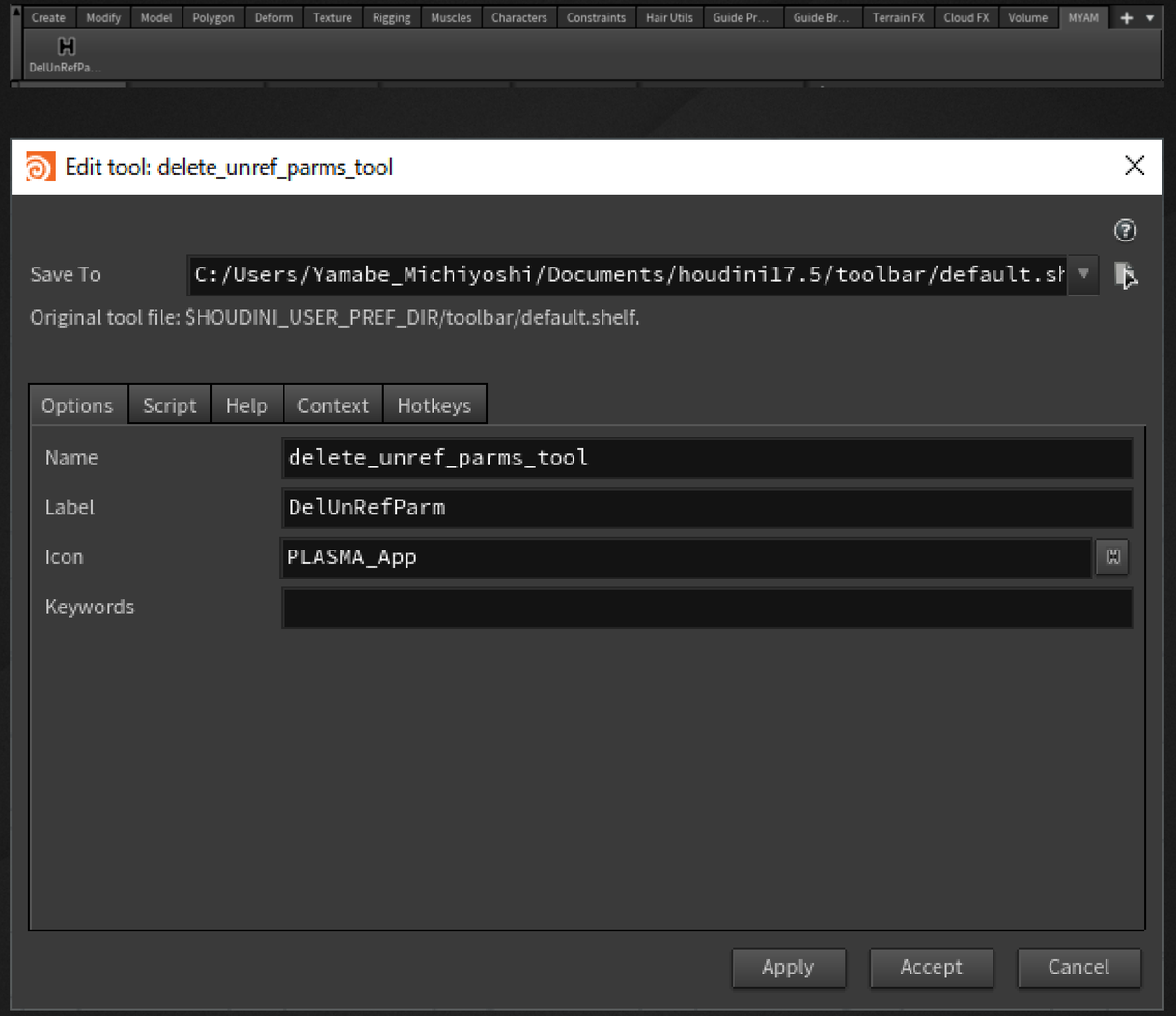
Python Source Editor

Shelf Tool

Python Node

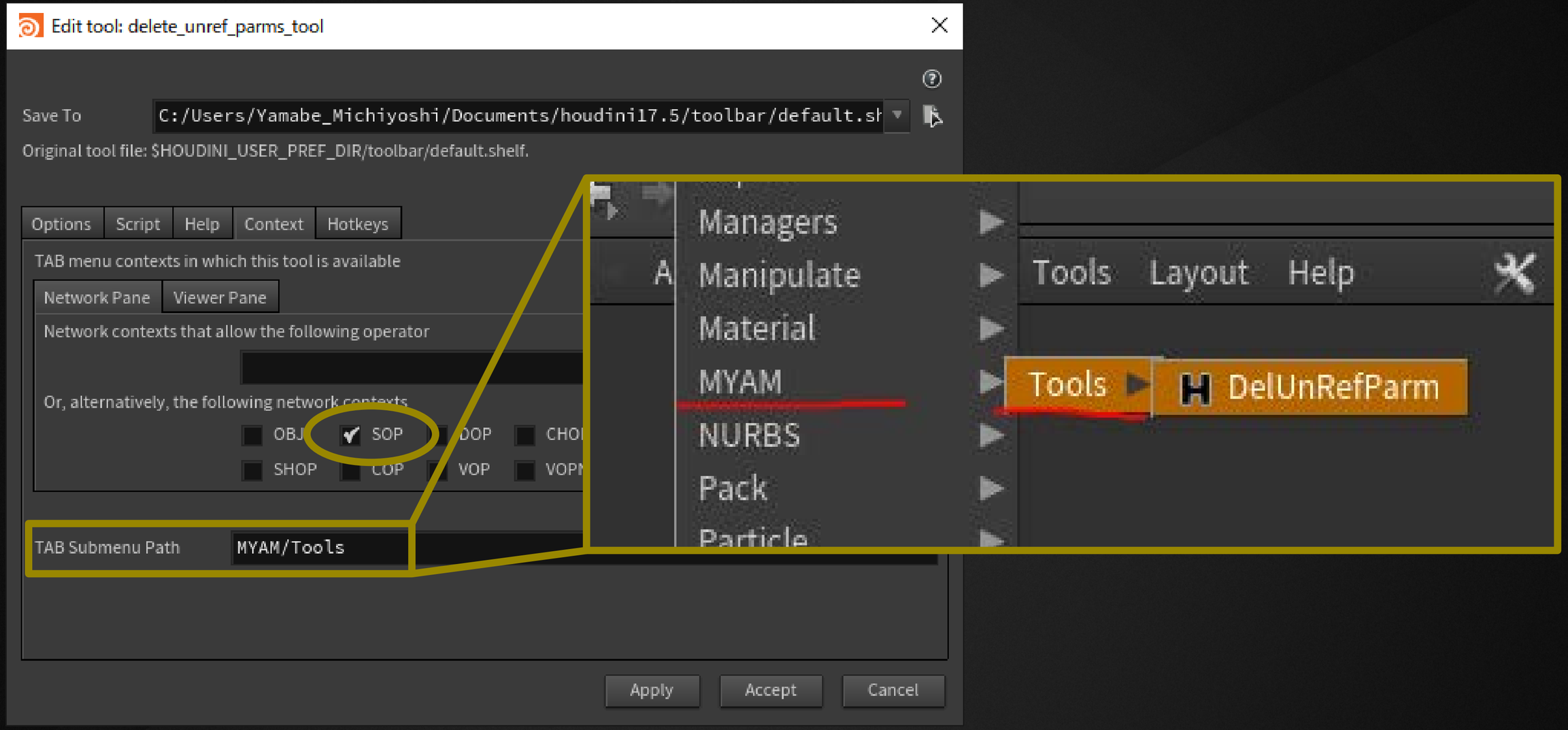
Parameter Callback Script

Shelf Tool



POLYPHONY™ DIGITAL

Shelf Tool



POLYPHONY DIGITAL

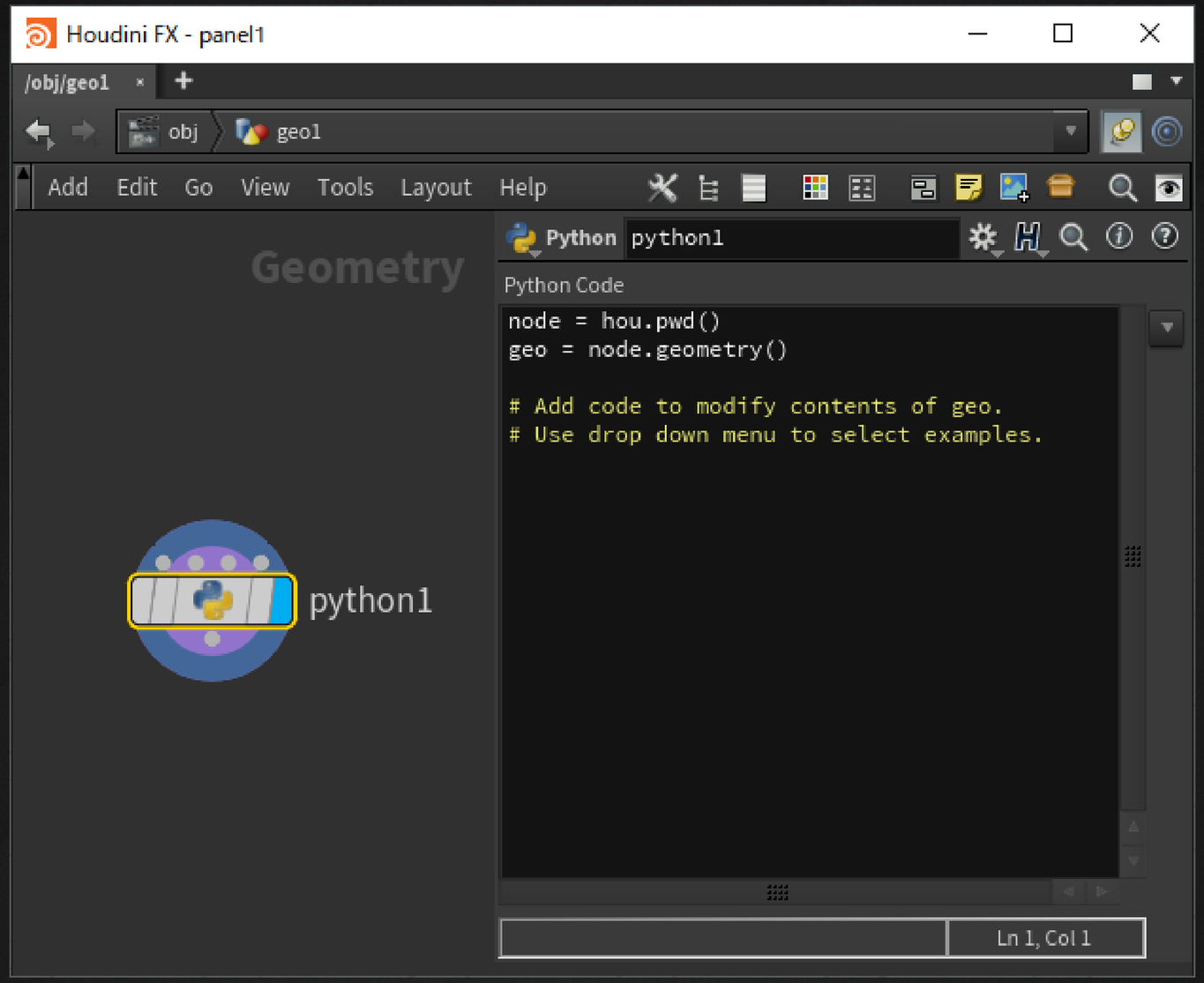
Python Source Editor

Shelf Tool

Python Node

Parameter Callback Script

Python Node



POLYPHONY™ DIGITAL

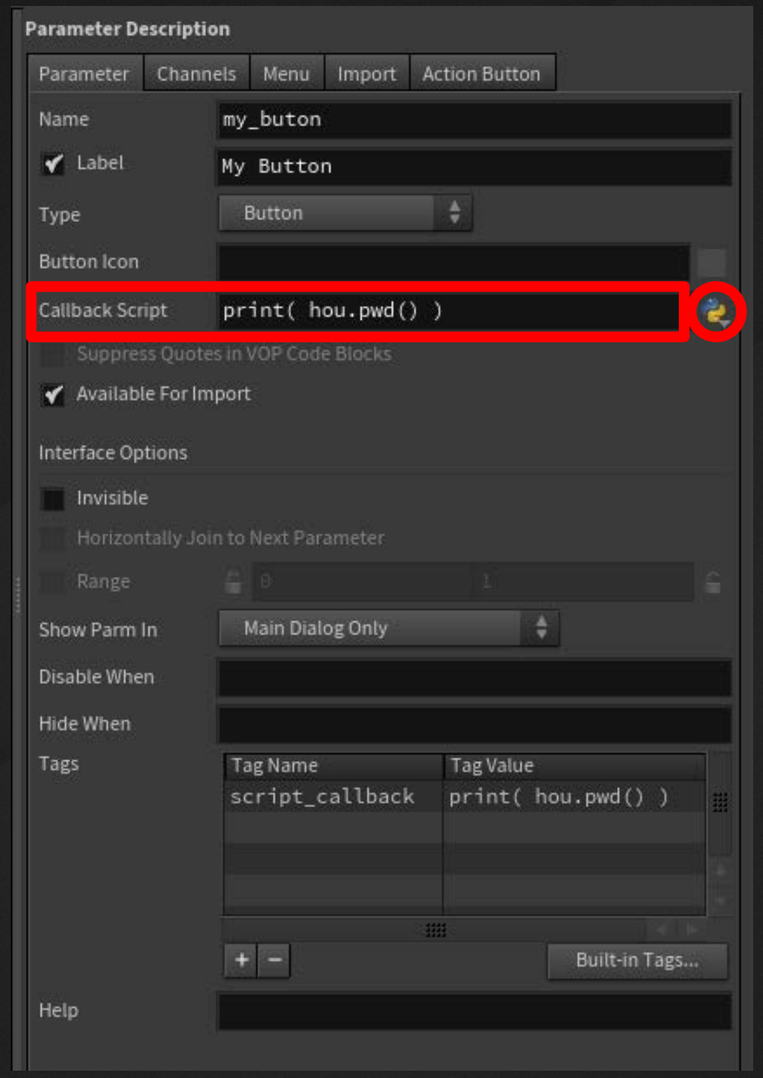
Python Source Editor

Shelf Tool

Python Node

Parameter Callback Script

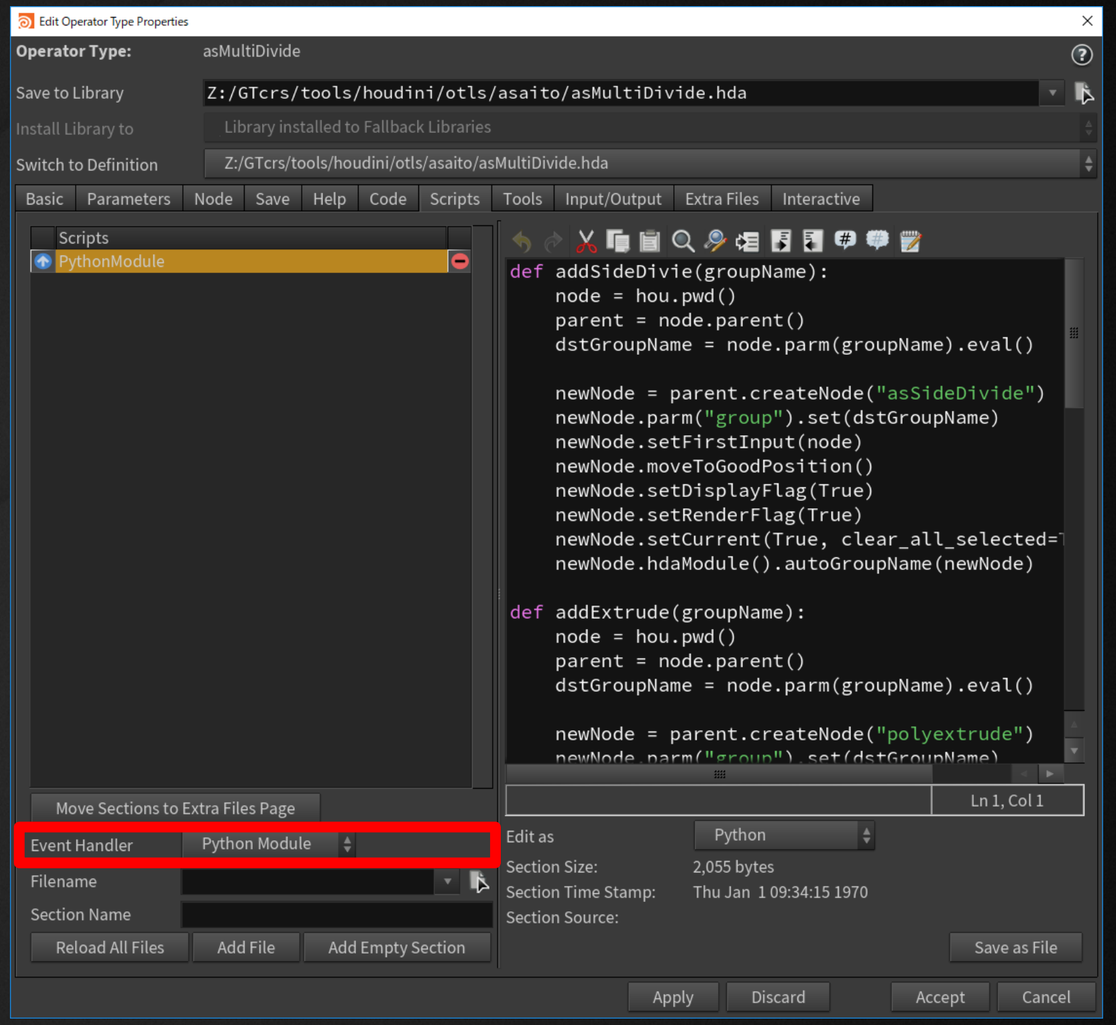
Parameter Callback Script



Parameter Callback Script (HDA埋め込みスクリプト)

- 埋め込みスクリプトはEvent HandlerをPython Modulesにすることで記述可能

- hou.pwd().hdaModule().member()
  - .pwd() = HDA自身
  - .hdaModule() = PythonModule
  - .member() = スクリプト内の関数



POLYPHONY DIGITAL

基本的なTIPSや注意点など

Qt for Pythonを利用する際のTIPS

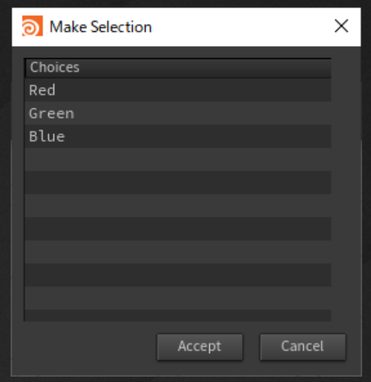
- ツールのGUIを表示するのに最低限必要な処理

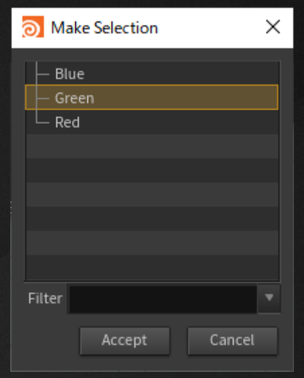
```
houdini_window = hou.qt.mainWindow()
tool_window = MyToolWindow( parent=houdini_window )
tool_window.setStyleSheet( hou.qt.styleSheet() )
tool_window.setProperty( "houdiniStyle", True )
```

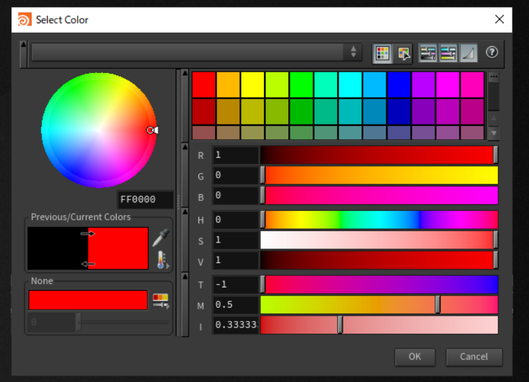
- QtDesighnerでGUIを作成後 pyside2-uic.exe を使用し .ui を .py に変換
  必須ではないが、自動補完機能に対応しているエディタで便利

```
pyside2-uic.exe –o my_gui.py my_gui.ui
```

その他便利な基礎知識や注意事項







- hou.ui モジュール
  HoudiniのGUI要素へのアクセス
  便利GUIを簡単作成

- hou.qt モジュール
  Qt for Pythonのクラスを継承しているカスタムWidgetクラスが多数収録

- utils系モジュール群
  内部的に利用されている便利モジュールが利用可能

  ~$HFS¥houdini¥python2.7libs

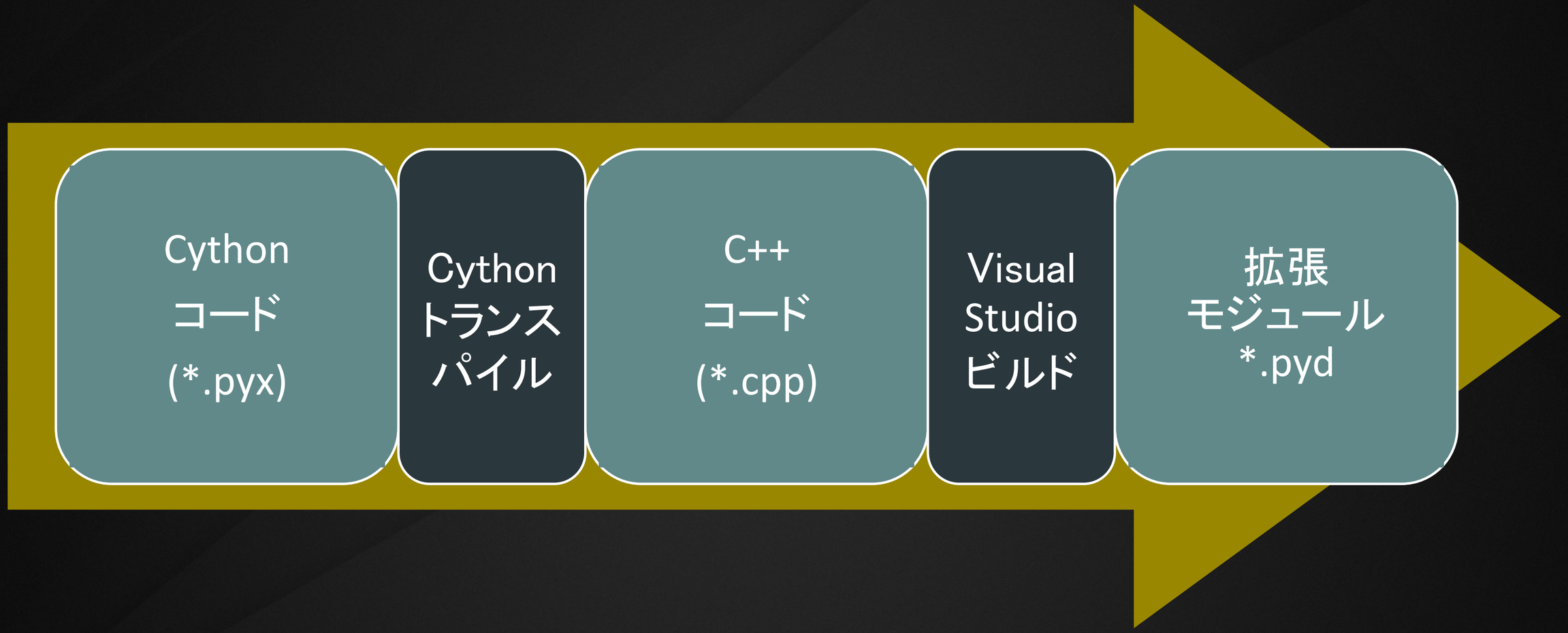
Houdini × Python 発展知識

Cythonによる高速化

Cythonとは

- C/C++とPythonのハイブリッドな言語
- Pythonに高速なC/C++の処理を組み込める
- C/C++ライブラリの使用も可能。もちろんHDKも可
- 事前に拡張モジュール(*.pyd)をビルドするので、ツールの配布が簡単

Cythonのビルドフロー



Cythonの利用法

- CythonはPython環境上にpipインストール

  pip install cython

  pythonフォルダ内にcython.exeが作成される

- Cythonコードは *.pyx で、*.cpp へのコンバートは cython.exe を使用

  cython.exe "cython_code.pyx" --cplus -o "cpp_code.cpp"

Cythonの効果

実行速度の検証(シンプルなループ)

- ループ回数＝100,000,000
- Pythonのみの場合
  6.815 [sec]
- Cythonで最適化した場合
  0.03 [sec]
- 約235倍の高速化

```
# シンプルなループ処理の速度比較
def loop_test_pure_python( loop_num ):
    res = 0

    for i in range(loop_num):
        res += i

    return res

def loop_test_full_cython( int loop_num ):
    cdef int i = 0
    cdef int res = 0

    for i in range(loop_num):
        res += i

    return res
```

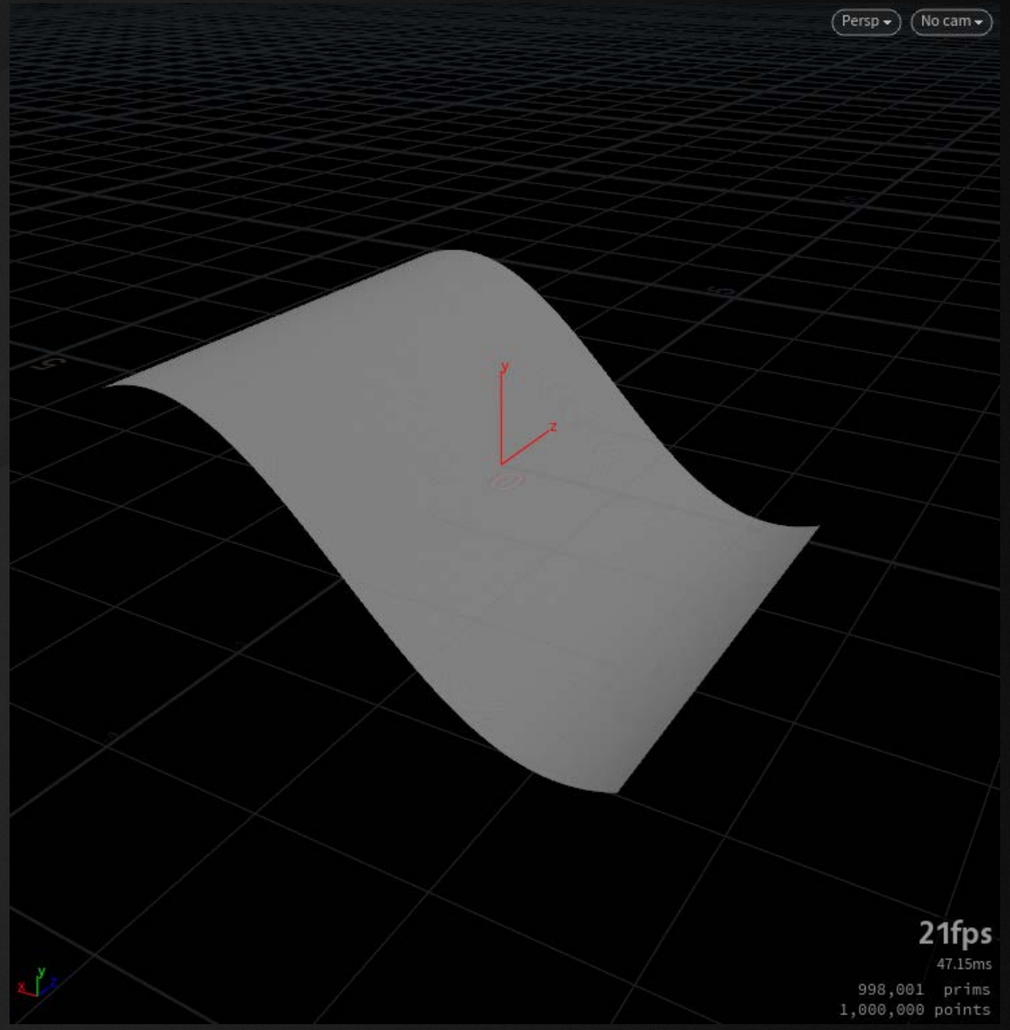
実行速度の検証（フィボナッチ数列の計算）

- ループ回数＝100,000,000
- Pythonのみの場合
  1.354 [sec]
- Cythonで最適化した場合
  0.028 [sec]
- 約48倍の高速化

```
# フィボナッチ数列
def fibonacci_pure_python( n ):
    a, b = 0.0, 1.0

    for i in range(n):
        a, b = a + b, a

    return a

def fibonacci_full_cython( int n ):
    cdef int a = 0
    cdef int b = 1

    for i in range(n):
        a, b = a + b, a

    return a
```

実行速度の検証（ジオメトリのデフォーム）

- 1,000,000ポイントのグリッドを変形
- 100フレームで要した時間を計測
- 速度比較はPythonに合わせ逐次実行で



Python per Points

Name	Cook Count	Cook (Avg)	Time
Total Statistics			14m 26.175s
Nodes			13m 36.785s
obj			13m 36.785s
grid_object1			13m 36.785s
python_houdini_per_points (S	99	8.250s	13m 36.785s

- 各速度テストはこの処理と同等の処理内容で行います

- 合計時間　816.785 s
- 平均時間　8.250 s

Wrangle (Details)

Name	Cook Count	Cook (Avg)	Time
Total Statistics			1m 05.383s
Nodes			50.047s
obj			50.047s
grid_object1			50.047s
wrangle_on_details (Sop/attri	99	0.000s	50.047s
attribvop1 (Sop/attribvop)	99	0.505s	50.019s

- 合計時間　50.019 s
- 平均時間　0.505 s

Cython × HDK

Name	Cook Count	Cook (Avg)	Time
Total Statistics			50.763s
Other			1.591s
Nodes			2.751s
obj			2.751s
grid_object1			2.751s
cython_on_points (Sop/pytho	99	0.028s	2.751s

- 合計時間　2.751 s
- 平均時間　0.028 s

番外編　Wrangle (Points)

Name	Cook Count	Cook (Avg)	Time
Total Statistics			17.315s
Nodes			0.659s
obj			0.659s
grid_object1			0.659s
wrangle_on_points (Sop/attri	99	0.000s	0.659s
attribvop1 (Sop/attribvop)	99	0.007s	0.648s

- Point Wrangleで並列実行 ＋ アトリビュートを直接読み書き

- 合計時間　0.648 s
- 平均時間　0.007 s

番外編 Python at Once

Name	Cook Count	Cook (Avg)	Time
Total Statistics			1m 36.858s
Nodes			47.354s
obj			47.354s
grid_object1			47.354s
python_houdini_at_once (Sop	99	0.478s	47.354s

- 逐次実行 ＋ 座標配列を一括読み書き
  hou.Node.pointFloatAttribValues() / .setPointFloatAttribValues()

- 合計時間　47.354s
- 平均時間　0.478 s

速度比較結果

	Python per Points	Wrangle (Detail)	Cython HDK	Wrangle (points)	Python at Once
合計時間	816.785 s	50.019 s	2.751 s	0.648 s	47.354 s
速度向上率	x 1	x 16	x 297	x 1260	x 17

	Python per Points	Wrangle (Detail)	Cython HDK	Wrangle (points)	Python at Once
平均時間	8.250 s	0.505s	0.028 s	0.007 s	0.478 s
速度向上率	x 1	x 16	x 295	x 1179	x 17

まとめ

- 最低限のノード・パラメータ操作知識だけでも便利なツールは作れる
  他のPythonツール作成経験があればすぐ着手可能

- Pythonはどうしても速度的に不利。高速化が必要。
  Cython × HDKを使うことで手軽に処理速度を大幅に改善できる

ご清聴ありがとうございました

質疑応答

付録

付録　参考資料

- hou package リファレンス
  https://www.sidefx.com/ja/docs/houdini/hom/hou/index.html
- Python スクリプトの場所
  https://www.sidefx.com/ja/docs/houdini/hom/locations.html
- ノード毎のユーザー定義データ
  https://www.sidefx.com/ja/docs/houdini/hom/nodeuserdata.html

付録　Houdini Pythonの考え方

Houdini内の要素は、全て「Pythonオブジェクト」として表現される

- hou.Node
- hou.NodeType
- hou.Parm
- hou.Geometry
- hou.Attrib
- hou.NetworkMovableItem
- hou.NodeConnection
- hou.Pane
- hou.PaneTab　など

付録　Houdini Pythonの考え方

要素のタイプは列挙型の値(enum値)で表現する事が多い

- hou.attribType.Point / Prim / Vertex / Global
- hou.filetype.Any / Image / Geometry / Hip　など
- hou.paneTabType.NetworkEditor / SceneViewer / MaterialPalette など

付録　ノードを探す関数、メソッド（一部）

- hou.node(node_path) -> hou.Node 指定したパスのノード

- hou.Node.parent() -> hou.Node 現在ノードを含むノード
- hou.Node.children() -> tuple of hou.Node 現在ノード直接内側にあるノード
- hou.Node.inputs() -> tuple of hou.Node 現在ノードの上流に接続されたノード
- hou.Node.outputs() -> tuple of hou.Node現在ノードの下流に接続されたノード
- hou.selectedNodes() -> tuple of hou.Node 選択中の全ノード

- hou.pwd() -> hou.Node　関数を呼び出した現行ノード
- hou.NetworkEditor.pwd().children() -> tuple of hou.Node　など

付録　パラメータを探す関数、メソッド（一部）

- hou.parm(parm_path) -> hou.Parm 指定したパスのパラメータ

- hou.Node.parm(parm_name) -> hou.Parm ノード内の指定パスのパラメータ
- hou.Node.parms() -> tuple of hou.Parm ノード内の全パラメータ
- hou.Node.spareParms() -> tuple of hou.Parm ノード内の全スペアパラメータ
- hou.Node.globParms(pattern) -> tuple of hou.Parm
  正規表現にマッチする全パラメータ

- hou.Parm.getReferencedParm() -> hou.Parm 参照しているパラメータ
- hou.Parm.parmsReferencingThis() -> tuple of hou.Parm　参照されている全パラメータ　など

付録　パラメータを操作する関数、メソッド（一部）

- hou.Parm.eval() -> 現在時間の値
- hou.Parm.set(value) -> 値をセットする
- hou.Parm.expression() -> str エクスプレッション文字列を取得
- hou.Parm.setExpression(expresstion_string) エクスプレッション文字列を設定
- hou.Node.setParms(parm_dict) -> パラメータ辞書をセット