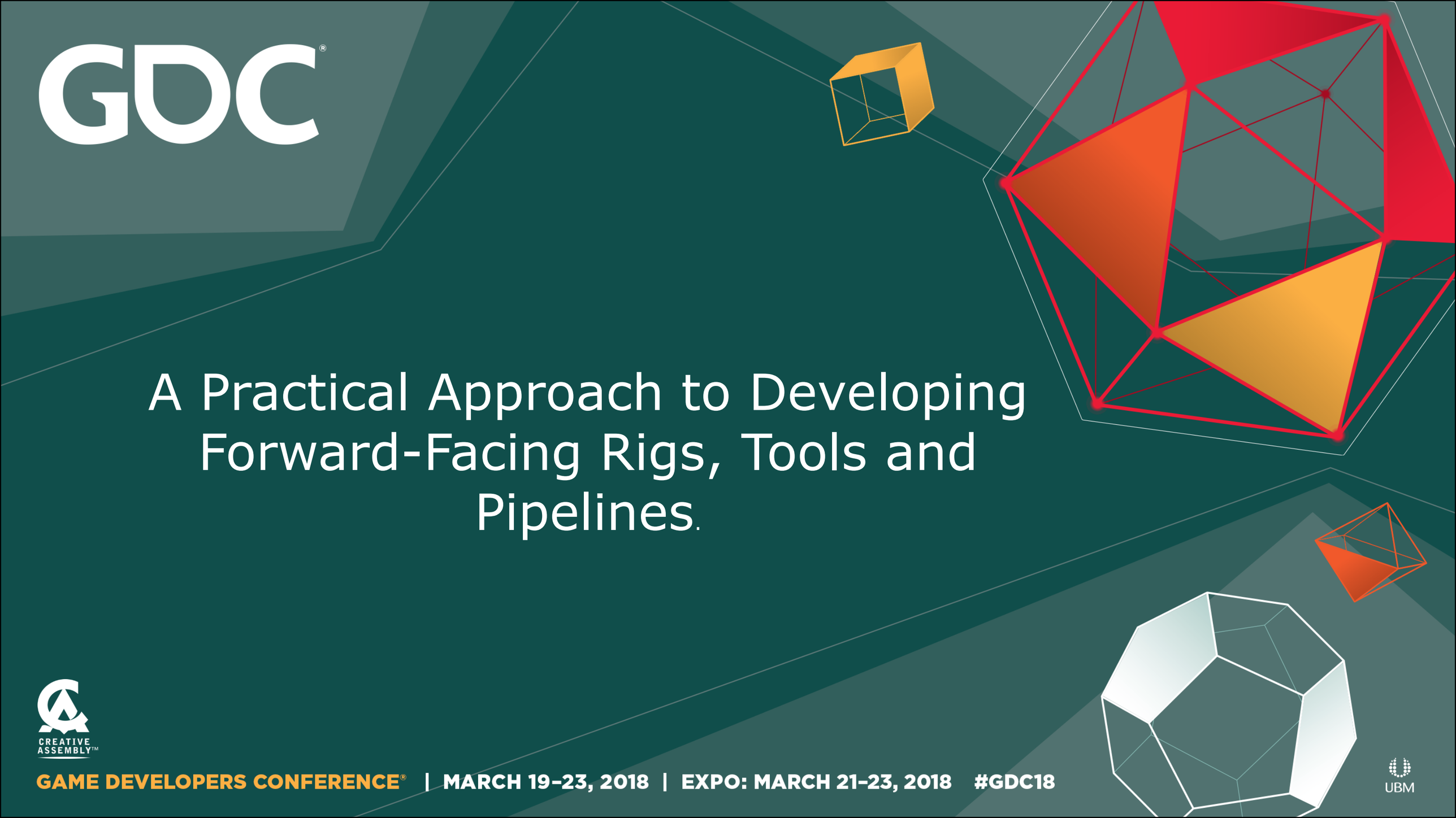
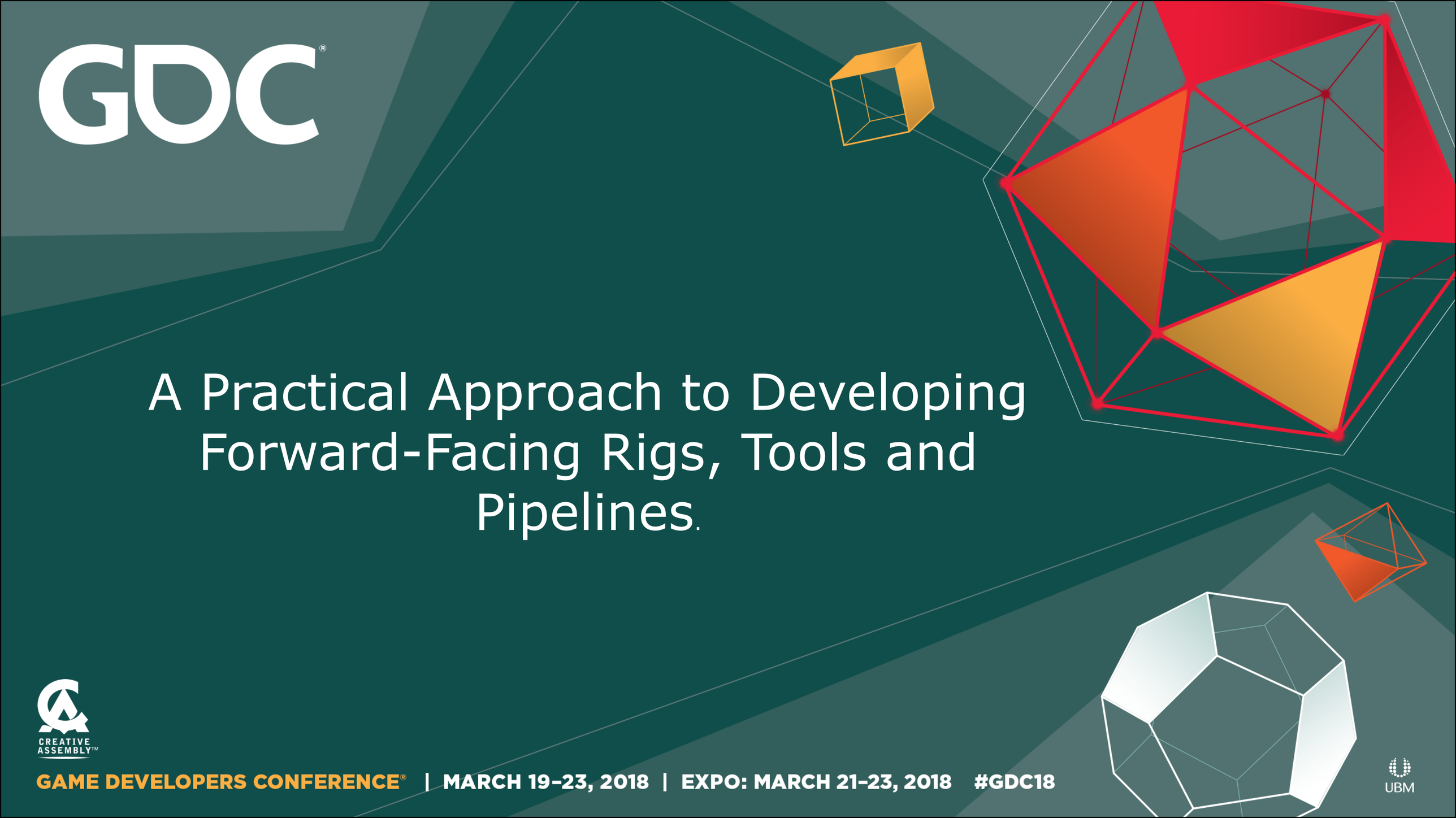
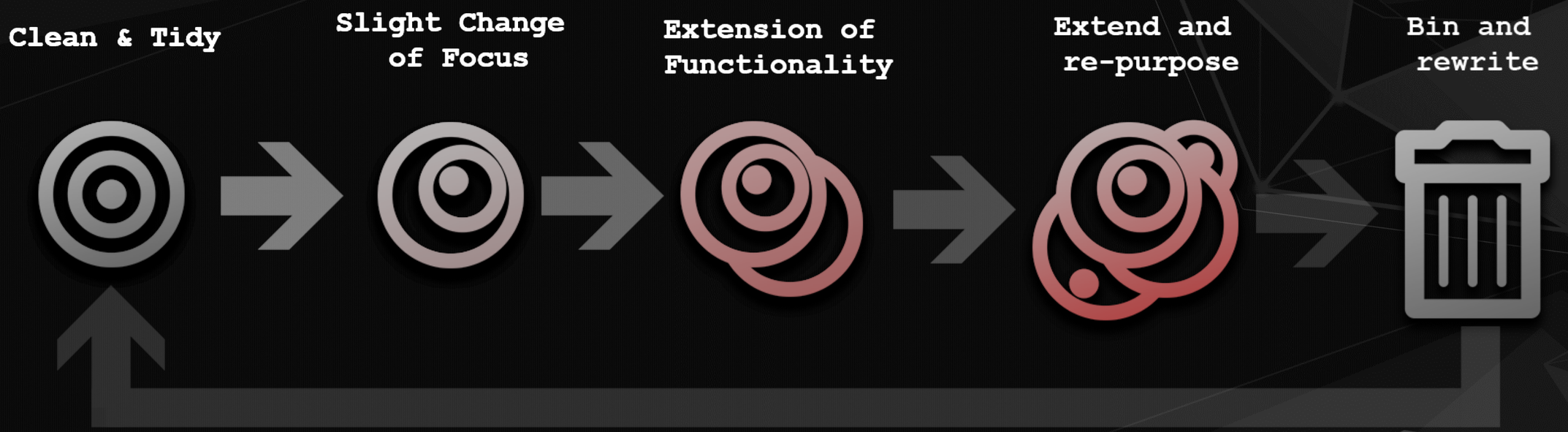

GDC
A Practical Approach to Developing Forward-Facing Rigs, Tools and Pipelines.
CREATIVE ASSEMBLY™
GAME DEVELOPERS CONFERENCE® | MARCH 19–23, 2018 | EXPO: MARCH 21–23, 2018 #GDC18
UBM


GDC
A Practical Approach to Developing Forward-Facing Rigs, Tools and Pipelines.
CREATIVE ASSEMBLY™
GAME DEVELOPERS CONFERENCE® | MARCH 19–23, 2018 | EXPO: MARCH 21–23, 2018 #GDC18
UBM


OUR
GAMES

The Problem: Changing requirements and the need for fast implementation



```python
# -- Collate a list of meshes to export
exportable_meshes = list()

# -- Only export skeletons if we find meshes with skin
# -- clusters
found_skins = False

# -- Find our exportable meshes
for mesh in pm.ls(type='mesh'):

    # -- Dont export meshes if they are not tagged as exportable
    if mesh.hasAttr('exportable') and mesh.exportable.get():
        exportable_meshes.append(mesh)

    # -- Check if this has a skin cluster attached
    if pm.mel.findRelatedSkinCluster(mesh.name()):
        found_skins = True

# -- Write them out
write_fbx(exportable_meshes)

# -- Now find our export roots
if found_skins:
    for joint in pm.ls(type='joint'):

        # -- Skip any non-root joints
        if isinstance(joint.getParent(), pm.nt.Joint):
            continue

        # -- Check if this exportable
        if joint.hasAttr('exportable') and joint.exportable.get():
            write_fbx(joint)
```

```
# -- Collate a list of meshes to export
exportable_meshes = list()

# -- Only export skeletons if we find meshes with skin
# -- clusters
found_skins = False

# -- Find our exportable meshes
for mesh in pm.ls(type='mesh'):

    # -- Dont export meshes if they are not tagged as exportable
    if mesh.hasAttr('exportable') and mesh.exportable.get():
        exportable_meshes.append(mesh)

    # -- Check if this has a skin cluster attached
    if pm.mel.findRelatedSkinCluster(mesh.name()):
        found_skins = True

# -- Write them out
write_fbx(exportable_meshes)

# -- Now find our export roots
if found_skins:
    for joint in pm.ls(type='joint'):

        # -- Skip any non-root joints
        if isinstance(joint.getParent(), pm.nt.Joint):
            continue

        # -- Check if this exportable
        if joint.hasAttr('exportable') and joint.exportable.get():
            write_fbx(joint)
```

```
# -- Collate a list of meshes to export
exportable_meshes = list()

# -- Only export skeletons if we find meshes with skin
# -- clusters
found_skins = False

# -- Find our exportable meshes
for mesh in pm.ls(type='mesh'):

    # -- Dont export meshes if they are not tagged as exportable
    if mesh.hasAttr('exportable') and mesh.exportable.get():
        exportable_meshes.append(mesh)

    # -- Check if this has a skin cluster attached
    if pm.mel.findRelatedSkinCluster(mesh.name()):
        found_skins = True

# -- Write them out
write_fbx(exportable_meshes)

# -- Now find our export roots
if found_skins:
    for joint in pm.ls(type='joint'):

        # -- Skip any non-root joints
        if isinstance(joint.getParent(), pm.nt.Joint):
            continue

        # -- Check if this exportable
        if joint.hasAttr('exportable') and joint.exportable.get():
            write_fbx(joint)
```

```
# -- Collate a list of meshes to export
exportable_meshes = list()

# -- Only export skeletons if we find meshes with skin
# -- clusters
found_skins = False

# -- Find our exportable meshes
for mesh in pm.ls(type='mesh'):

    # -- Skip any non-exportable meshes
    if not mesh.hasAttr('exportable') or not mesh.exportable.get():
        continue

    # -- Collate the mesh for exporting to fbx
    exportable_meshes.append(mesh)

    # -- Check if this has a skin cluster attached
    if pm.mel.findRelatedSkinCluster(mesh.name()):
        found_skins = True

    # -- Check if the mesh has cloth applied to it
    if mesh.inputs(type='customClothSolver'):
        write_cloth(
            mesh,
            find_affecting_bones(mesh)
        )

# -- Write them out
write_fbx(exportable_meshes)

# -- Now find our export roots
if found_skins:
    for joint in pm.ls(type='joint'):

        # -- Skip any non-root joints
        if isinstance(joint.getParent(), pm.nt.Joint):
            continue

        # -- Check if this exportable
        if joint.hasAttr('exportable') and joint.exportable.get():
            write_fbx(joint)
```

```
# -- Find our exportable meshes
for mesh in pm.ls(type='mesh'):

    # -- Skip any non-exportable meshes
    if not mesh.hasAttr('exportable') or not mesh.exportable.get():
        continue

    # -- Collate the mesh for exporting to fbx
    exportable_meshes.append(mesh)

    # -- Check if this has a skin cluster attached
    if pm.mel.findRelatedSkinCluster(mesh.name()):
        found_skins = True

    # -- Check if the mesh has cloth applied to it
    if mesh.inputs(type='customClothSolver'):
        write_cloth(
            mesh,
            find_affecting_bones(mesh)
        )
```

```
# -- Store a list of meshes that have vertex data
# -- relating to runtime rigging
runtime_meshes = list()
```

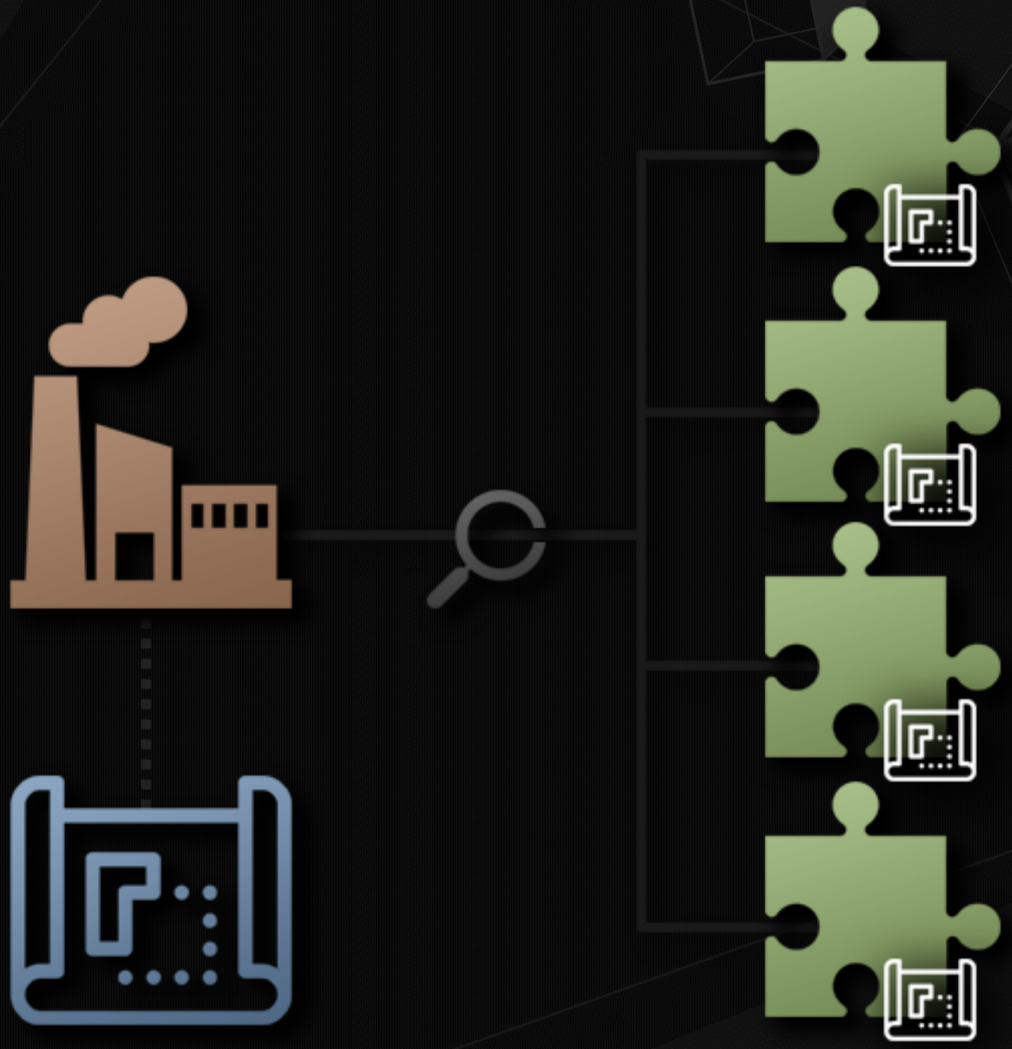
```
        # -- Write our runtime rig data if we're a skinned mesh
        if mesh.hasAttr('runtimeVertexData'):
            runtime_meshes.append(mesh)
```

```
# -- Store list of runtime bones
runtime_bones = list()
```

```
        if joint.hasAttr('runtimeData'):
            runtime_bones.append(joint)
```

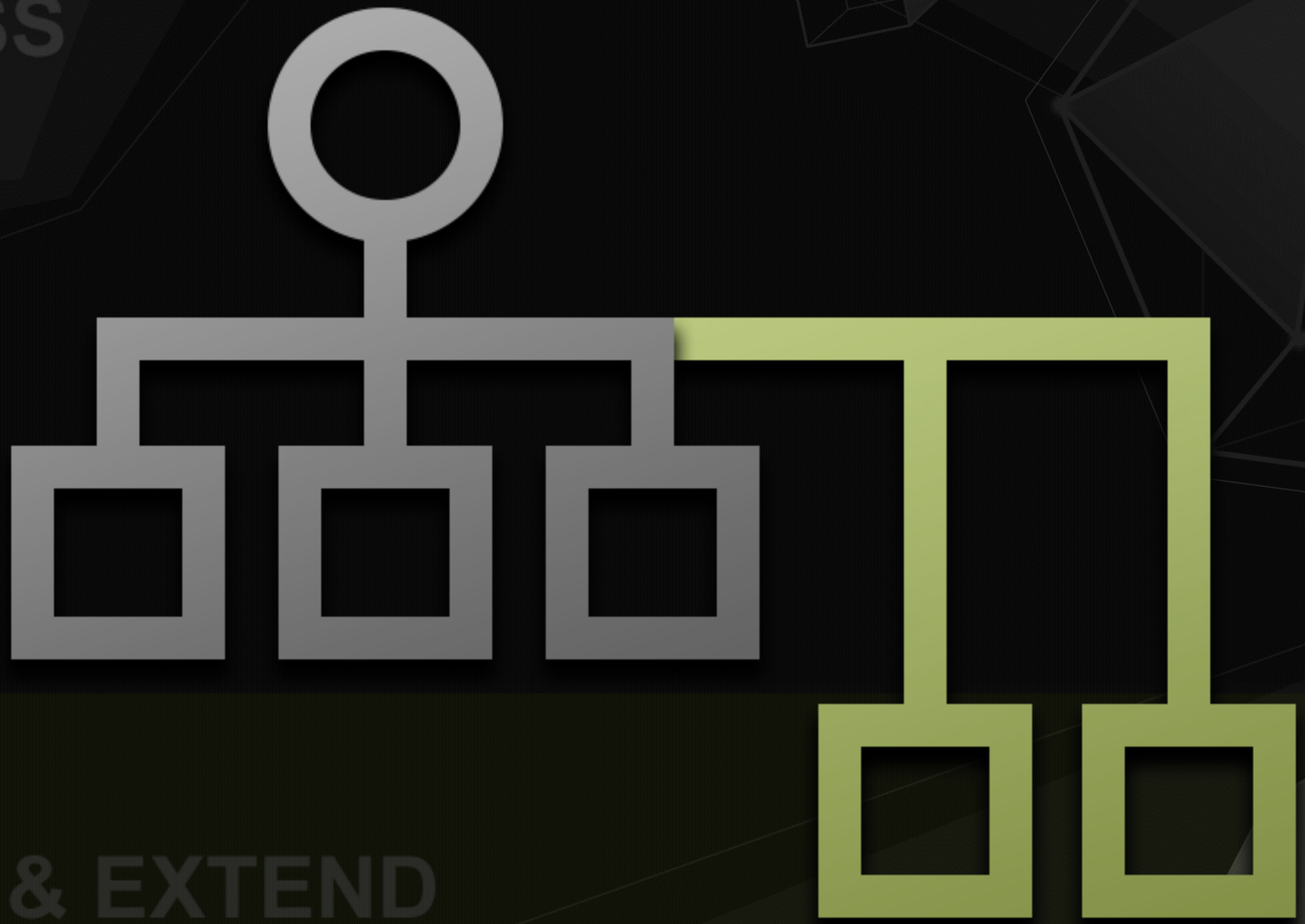
```
# -- Export any runtime data
if runtime_meshes and runtime_bones:
    write_vertex_attachment_data(
        runtime_meshes + runtime_bones
    )
```

https://sourcemaking.com/design_patterns



CREATIVE ASSEMBLY™

BASE CLASS



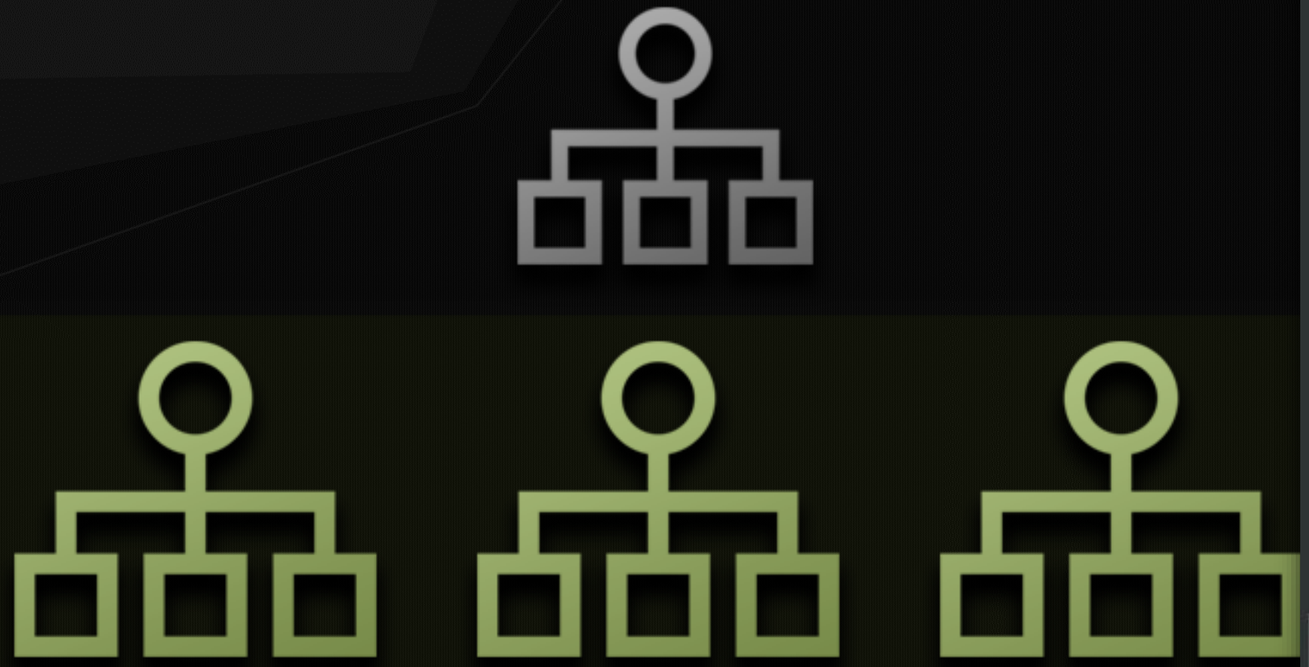
SUBCLASS & EXTEND

An abstract class…

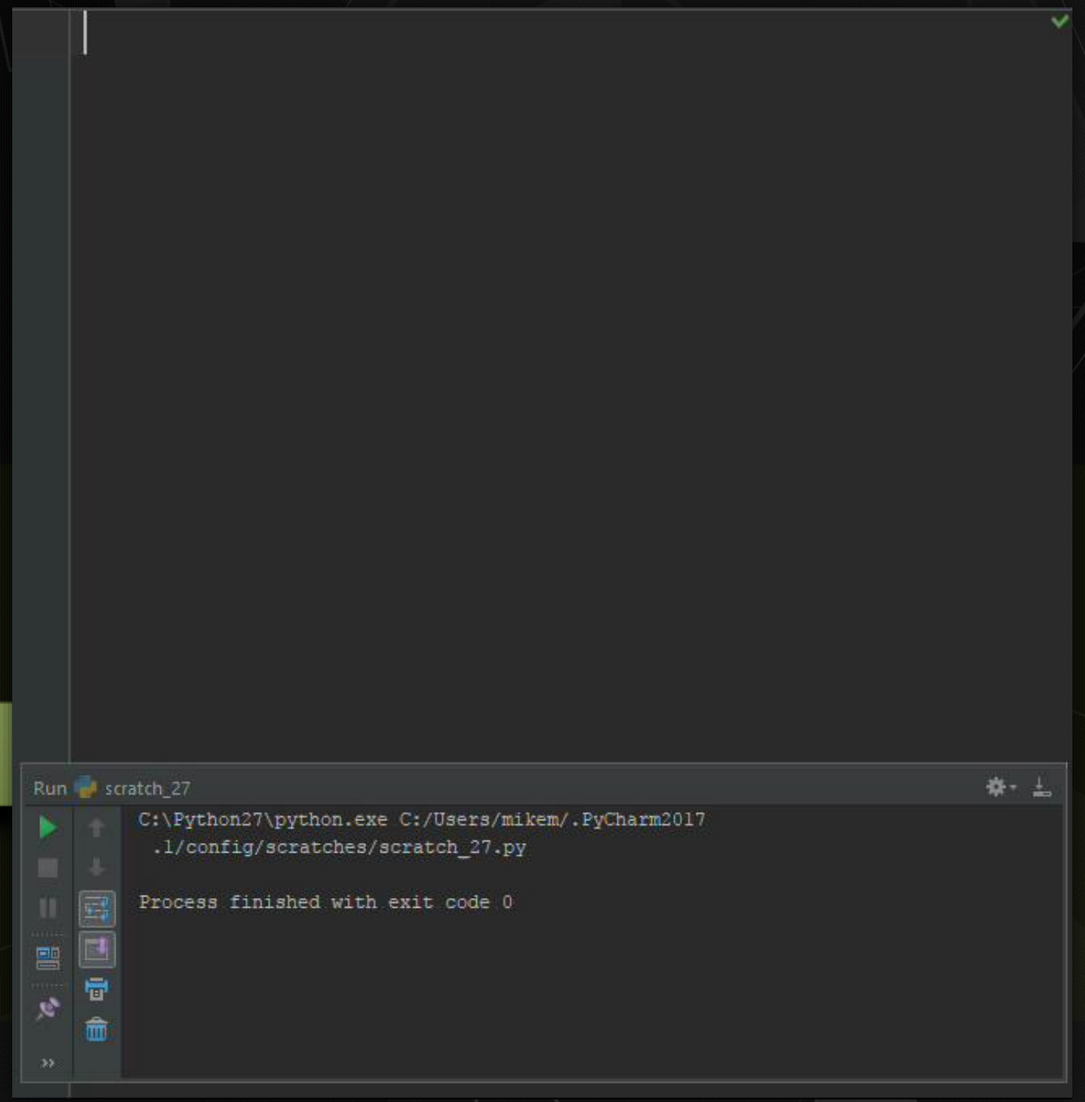
- Can never be instanced directly
- Contains no functionality
- Must have all methods re-implemented by subclasses

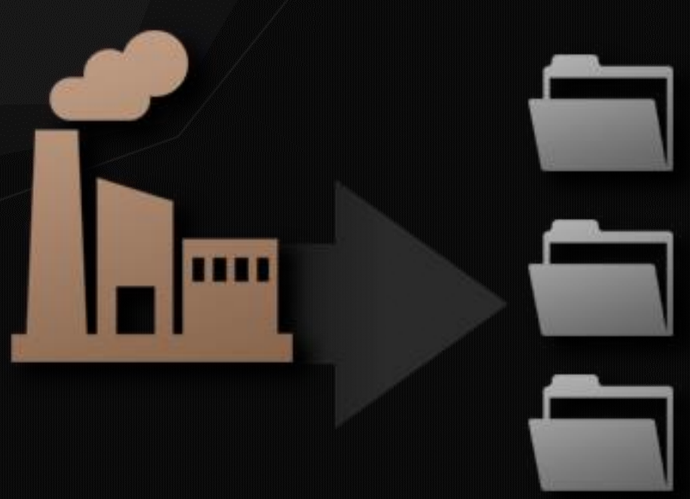
CREATIVE ASSEMBLY™

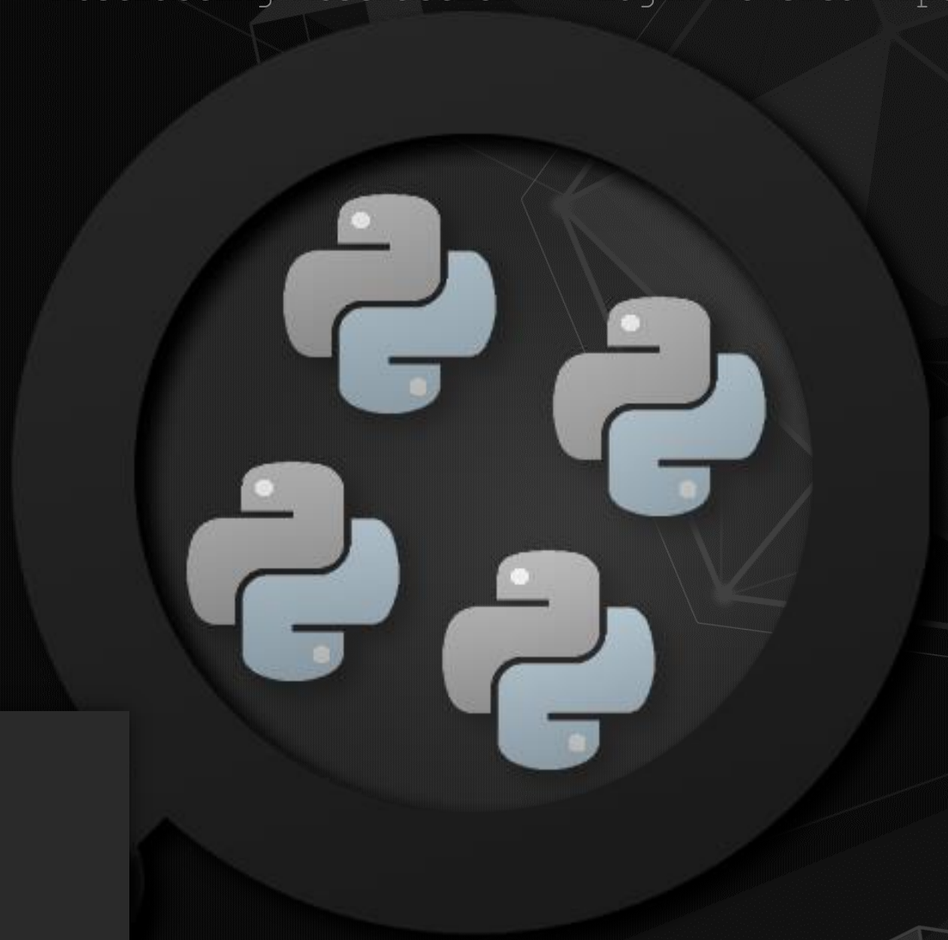
ABSTRACT



SUBCLASS & IMPLEMENT







```python
# -- Cycle the available plugins
for plugin_name in factory.available():

    # -- Get the plugin
    plugin = factory.get(plugin_name)

    # -- Instance the plugin and do something with
    # -- it
    plugin().do_something()
```

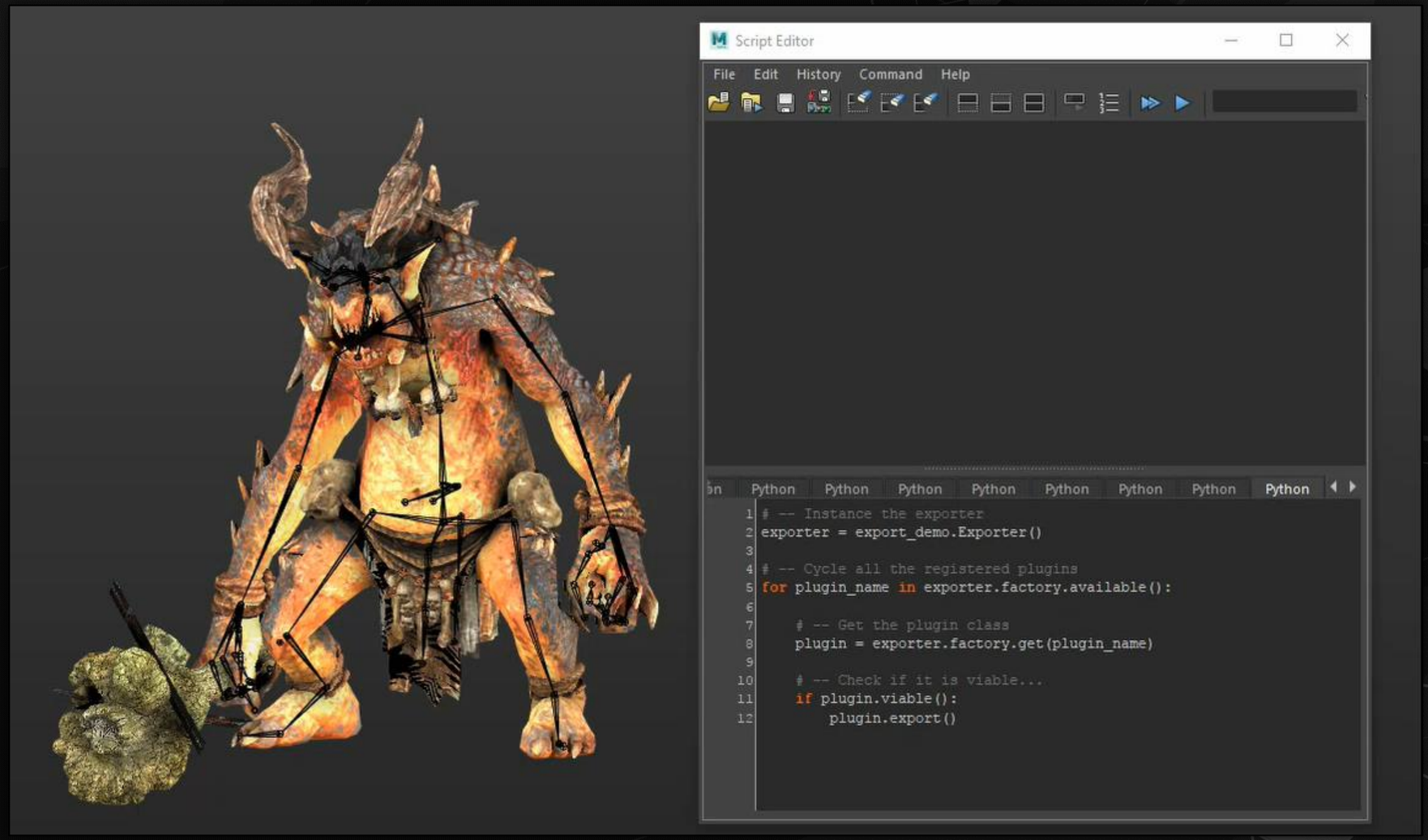
```python
# -- Cycle all locations recursively
for root, _, files in os.walk(directory):

    ...
    # -- Cycle all the items in the module
    for item in dir(py_module):

        ...
        # -- If it is a subclass of the defined abstract
        # -- register it (store the plugin)
        if issubclass(item, self._expected_base):
            self.register(item)
```

```python
class ExportPlugin(object):

    # -- Unique identifier to differentiate this plugin
    # -- from any other plugin
    Identifer = ''

    @classmethod
    def viable(cls, **kwargs):
        """
        Should test for anything in the scene/context which the
        plugin can extract

        :param context: If None, the entire scene is searched, otherwise
            any nodes under the contextual node will be searched

        :return: True if anything is found
        """
        return False

    @classmethod
    def export(cls, directory, **kwargs):
        """
        This will export any data out to external files

        :param directory: Location to store/save data
        :param context: Context to search within
        :param kwargs: Optional arguments to pass to the plugin

        :return: List of files exported
        """
        return list()
```

```
PluggedFactory(object)
    __init__(self, name_attr='name', version_attr, env_vars, base_class, paths)
    available(self)
    available_versions(self, plugin_name)
    clear_plugins(self)
    get(self, plugin_name, version=-1, refresh=False)
    paths(self)
    plugins(self)
    refresh(self)
    register(self, plugin_class)
    register_paths(self, paths, refresh=False)
```

```python
class ExportPlugin(object):

    # -- Unique identifier to differentiate this plugin
    # -- from any other plugin
    Identifer = 'Geometry'

    @classmethod
    def viable(cls, **kwargs):
        return bool(cls._exportables())

    @classmethod
    def export(cls, directory, **kwargs):
        for mesh in cls._exportables():
            write_fbx(
                mesh,
                os.path.join(
                    directory,
                    mesh.name(),
                ),
            )

    @classmethod
    def _exportables(cls):

        # -- Find all meshes which may be exportable
        exportables = list(
            maya.cmds.ls(
                '*.exportabe',
                objects=True,
                recursive=True,
                type='mesh',
            )
        )

        # -- Return True as soon as we hit an exportable mesh
        for exportable in exportables:
            if maya.cmds.getattr(exportable + '.exportable'):
                yield exportable
```

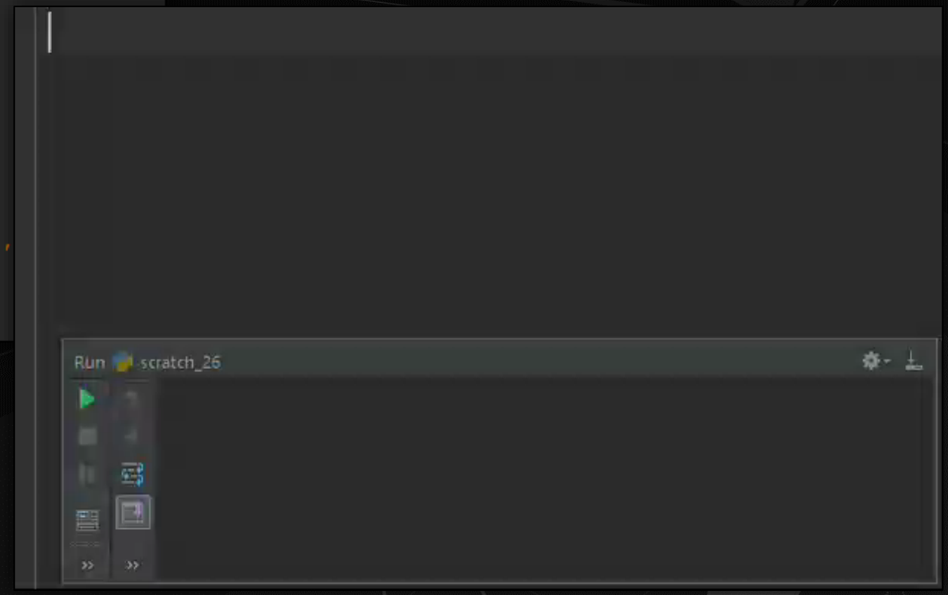
```python
class ExportPlugin(object):

    # -- Unique identifier to differentiate this plugin
    # -- from any other plugin
    Identifer = 'Skeleton'

    @classmethod
    def viable(cls, **kwargs):
        return bool(cls._exportables())

    @classmethod
    def export(cls, directory, **kwargs):
        joints = list(cls._exportables())

        write_fbx(
            joints,
            os.path.join(
                directory,
                joints[0].getParent(-1).name(),
            ),
        )

    @classmethod
    def _exportables(cls):

        # -- Find all meshes which may be exportable
        exportables = list(
            maya.cmds.ls(
                '*.exportable',
                objects=True,
                recursive=True,
                type='joint',
            )
        )

        # -- Return True as soon as we hit an exportable mesh
        for exportable in exportables:
            if maya.cmds.getattr(exportable + '.exportable'):
                yield exportable
```

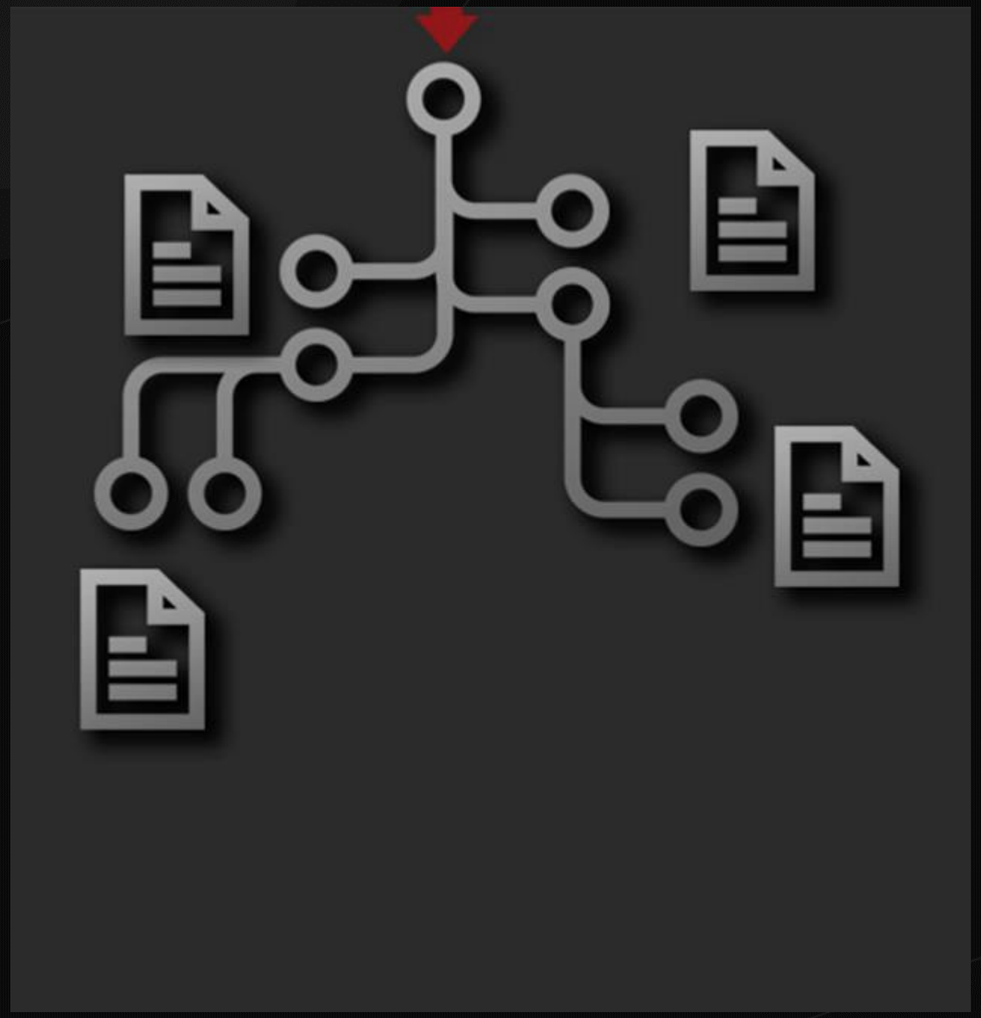

Script Editor
File Edit History Command Help
Python Python Python Python Python Python Python Python
# -- Instance the exporter
exporter = export_demo.Exporter()
# -- Cycle all the registered plugins
for plugin_name in exporter.factory.available():
# -- Get the plugin class
plugin = exporter.factory.get(plugin_name)
# -- Check if it is viable...
if plugin.viable():
plugin.export()

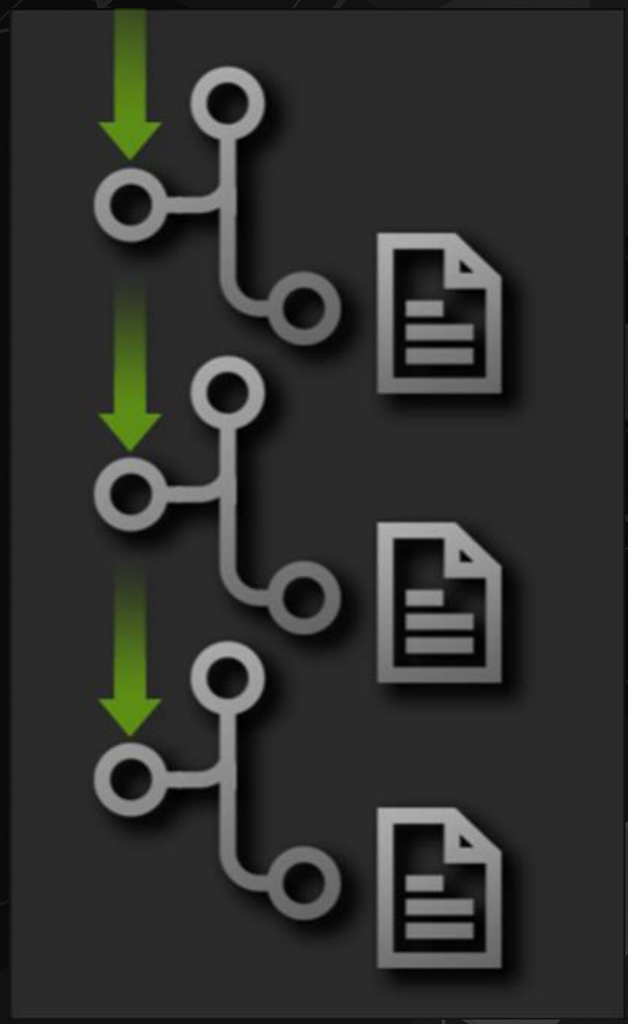

Geometry
Meshes
Cloth
Metadata

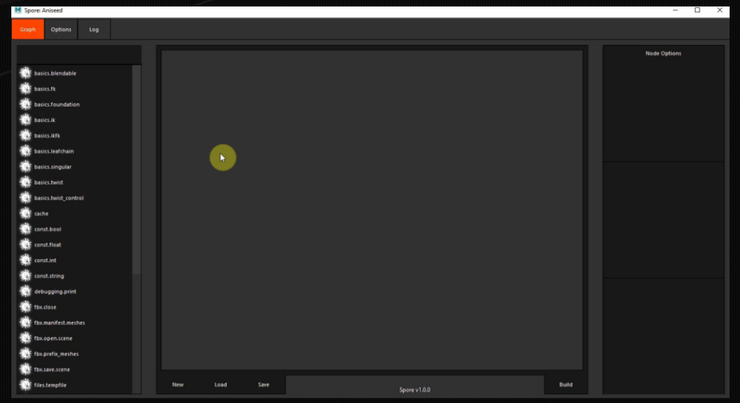
```python
def export_scene(directory, **kwargs):

    # -- Cycle all our export plugins
    for plugin_type in factory.available():

        # -- Get the plugin class
        plugin = factory.get(plugin_type)

        # -- Check if the plugin has anything it can operate
        # -- on
        if plugin.viable(context=None, **kwargs):
            plugin.export(
                directory,
                context=None,
                **kwargs
            )


# -- Call the export, passing a skin-specific option
export_scene(
    'c:/some/dir',
    bones_per_vertex_limit=4,
)
```

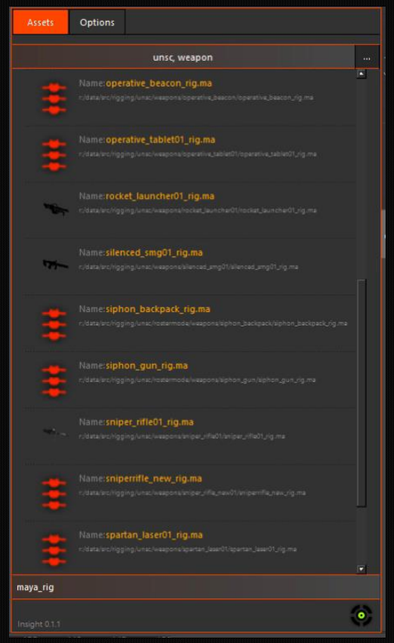
```python
class ExportPlugin(object):

    Identifer = 'Skeleton'

    @classmethod
    def export(cls, directory, context=None, **kwargs):

        joints = list(cls._exportables())

        # -- Limit our bones per vertex to 4, or a value
        # -- specified by the arguments
        limit_bones_per_vertex(
            kwargs.get(
                'limit_bones_per_vertex',
                4,
            ),
            joints,
        )
        write_fbx(
            joints,
            os.path.join(
                directory,
                joints[0].getParent(-1).name(),
            ),
        )
```

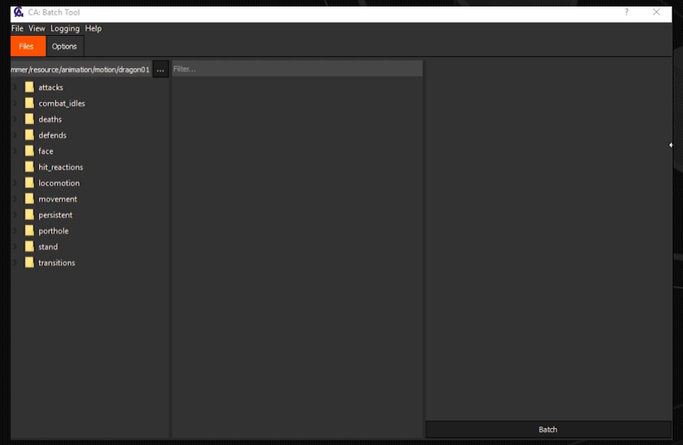


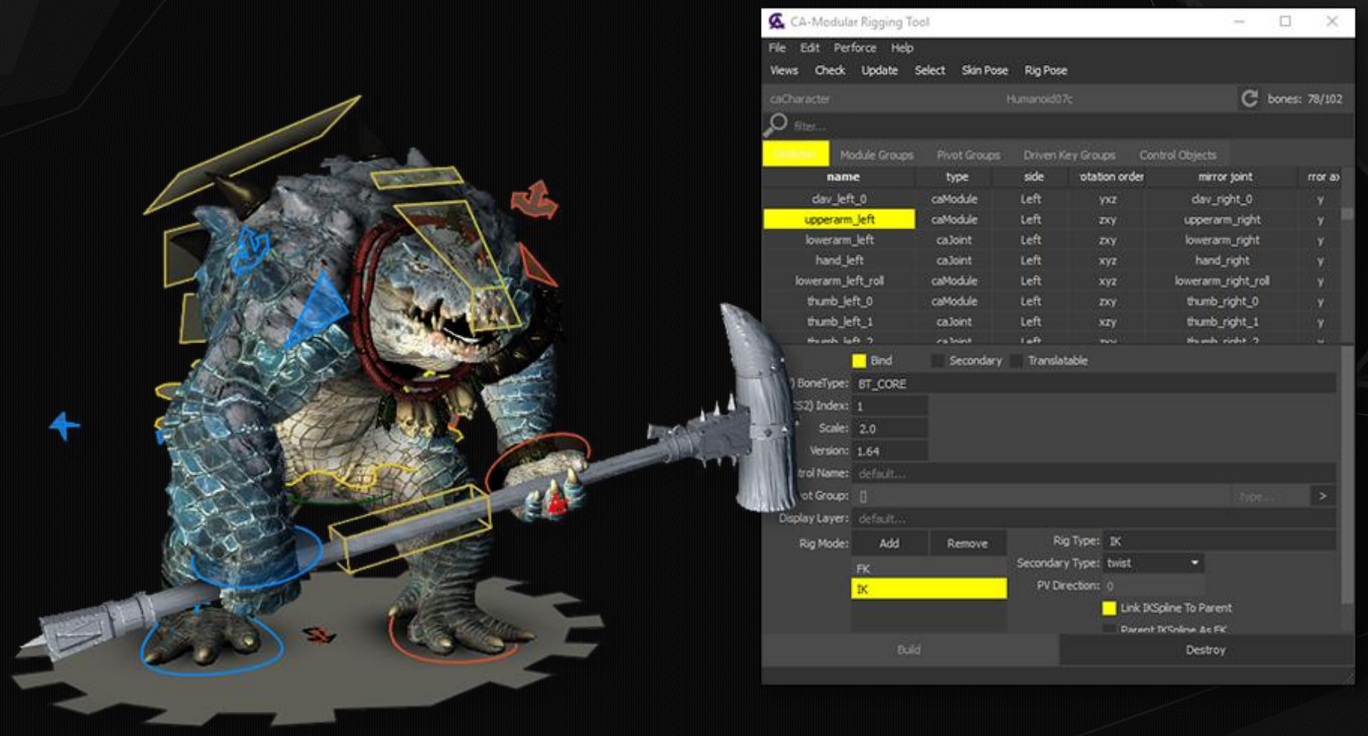


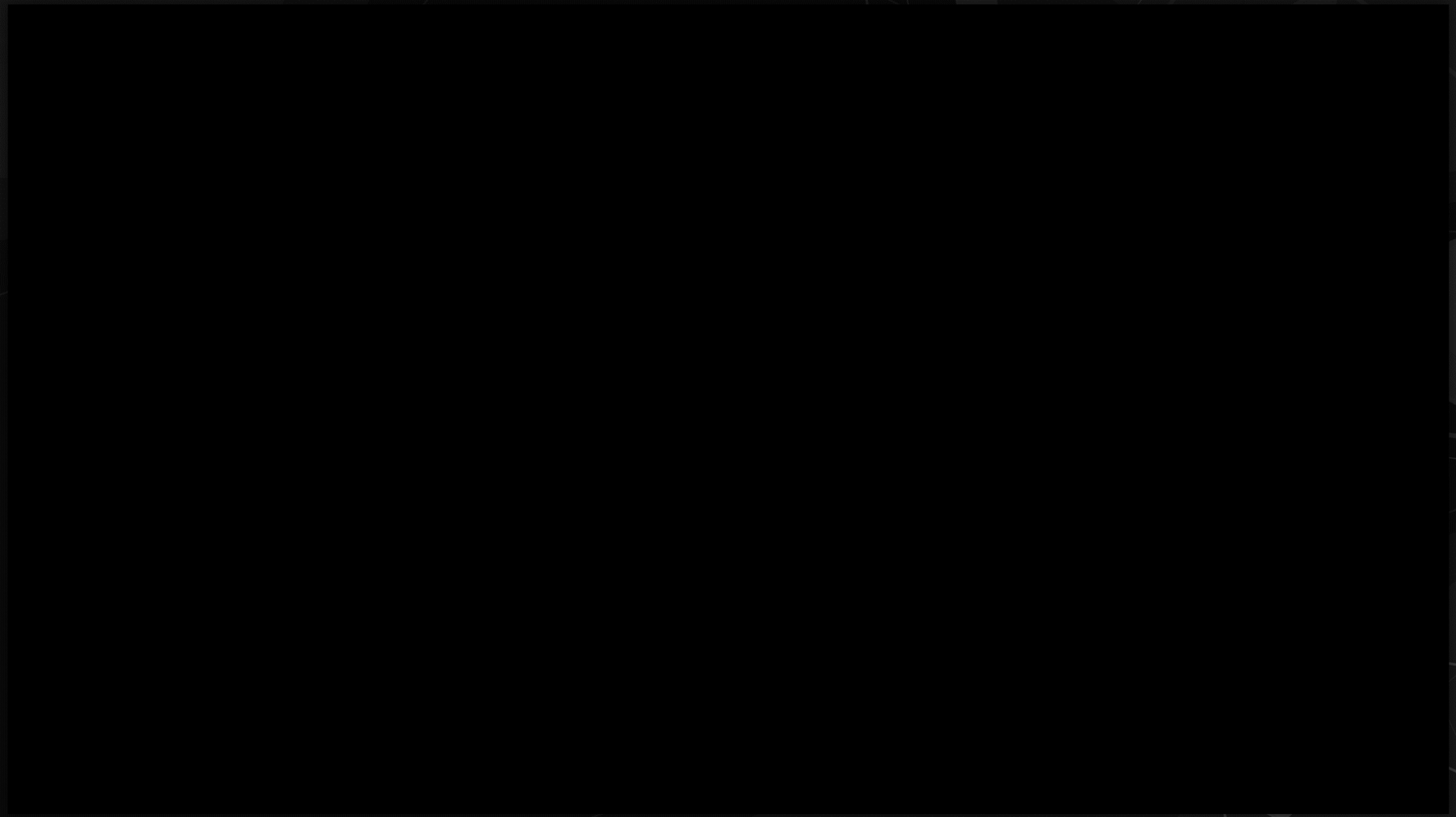


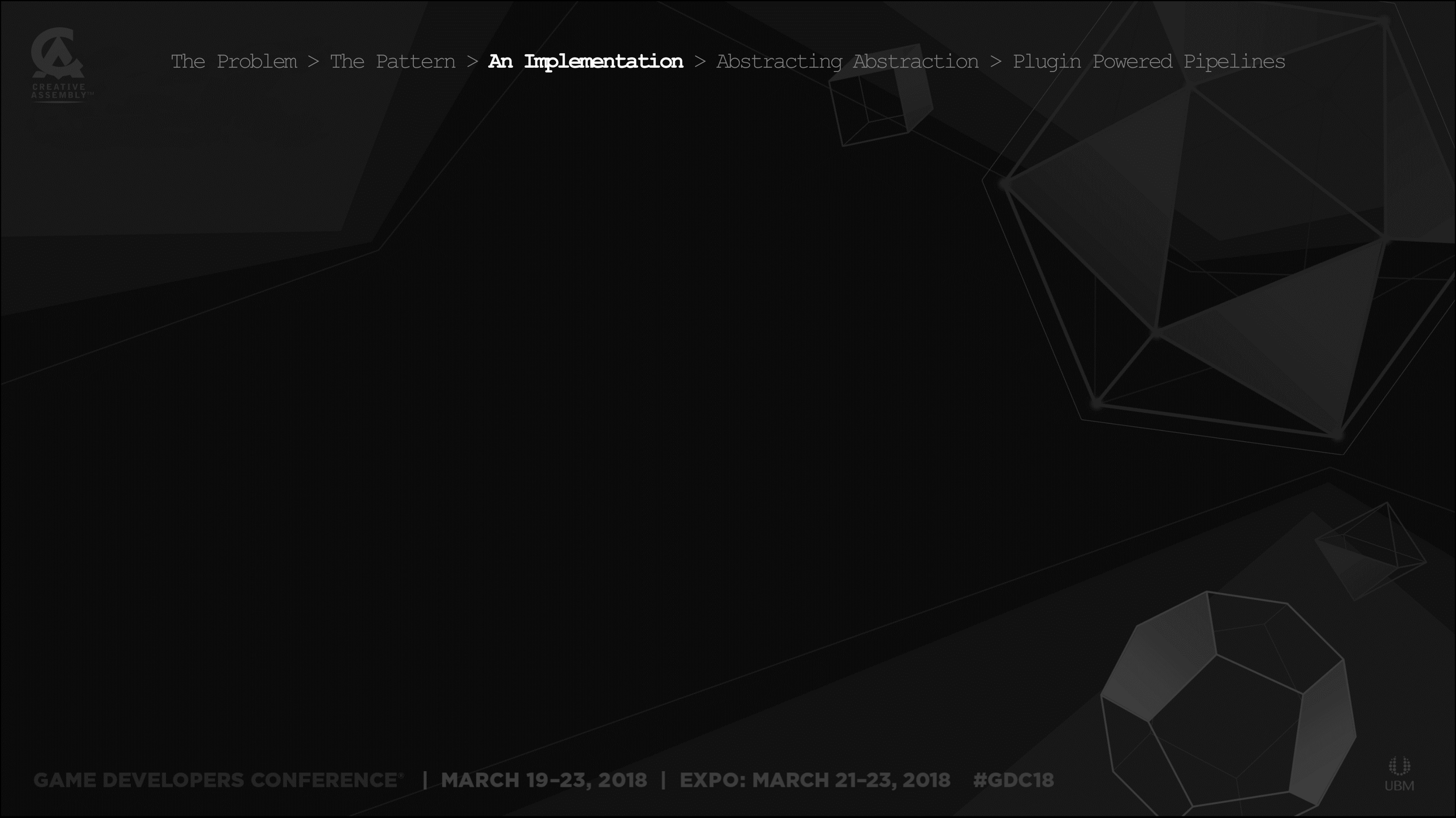

Spore: Aniseed
Graph
Options
Log
basics.blendable
basics.fk
basics.foundation
basics.ik
basics.ikfk
basics.leafchain
basics.singular
basics.twist
basics.twist_control
cache
const.bool
const.float
const.int
const.string
debugging.print
fbx.close
fbx.manifest.meshes
fbx.open.scene
fbx.prefix_meshes
fbx.save.scene
files.tempfile
Node Options
New
Load
Save
Spore v1.0.0
Build


Assets
Options
unsc, weapon
Name:operative_beacon_rig.ma
Name:operative_tablet01_rig.ma
Name:rocket_launcher01_rig.ma
Name:silenced_smg01_rig.ma
Name:siphon_backpack_rig.ma
Name:siphon_gun_rig.ma
Name:sniper_rifle01_rig.ma
Name:sniperrifle_new_rig.ma
Name:spartan_laser01_rig.ma
maya_rig
Insight 0.1.1


CA: Batch Tool
File
View
Logging
Help
Files
Options
attacks
combat_idles
deaths
defends
face
hit_reactions
locomotion
movement
persistent
porthole
stand
transitions
Batch


CA-Modular Rigging Tool
File Edit Perforce Help
Views Check Update Select Skin Pose Rig Pose
caCharacter
Humanoid07c
bones: 78/102
filter...
Module Groups
Pivot Groups
Driven Key Groups
Control Objects
name
type
side
otation order
mirror joint
clav_left_0
caModule
Left
yxz
clav_right_0
upperarm_left
caModule
Left
zxy
upperarm_right
lowerarm_left
caJoint
Left
zxy
lowerarm_right
hand_left
caJoint
Left
xyz
hand_right
lowerarm_left_roll
caModule
Left
xyz
lowerarm_right_roll
thumb_left_0
caModule
Left
zxy
thumb_right_0
thumb_left_1
caJoint
Left
xzy
thumb_right_1
Bind
Secondary
Translatable
BoneType:
BT_CORE
Index:
1
Scale:
2.0
Version:
1.64
Rig Mode:
Add
Remove
FK
IK
Rig Type:
IK
Secondary Type:
twist
PV Direction:
Link IKSpline To Parent
Build
Destroy





Growing list of rig requirements:

Standard IK

Spline IK

Optional Ankle Control for >2 bone ik

Standard or Reversed Pole Vector

Ability to rebuild consistently (backward compat)

Stretch

Growing list of rig requirements:

Standard IK

Spline IK

Optional Ankle Control for >2 bone ik

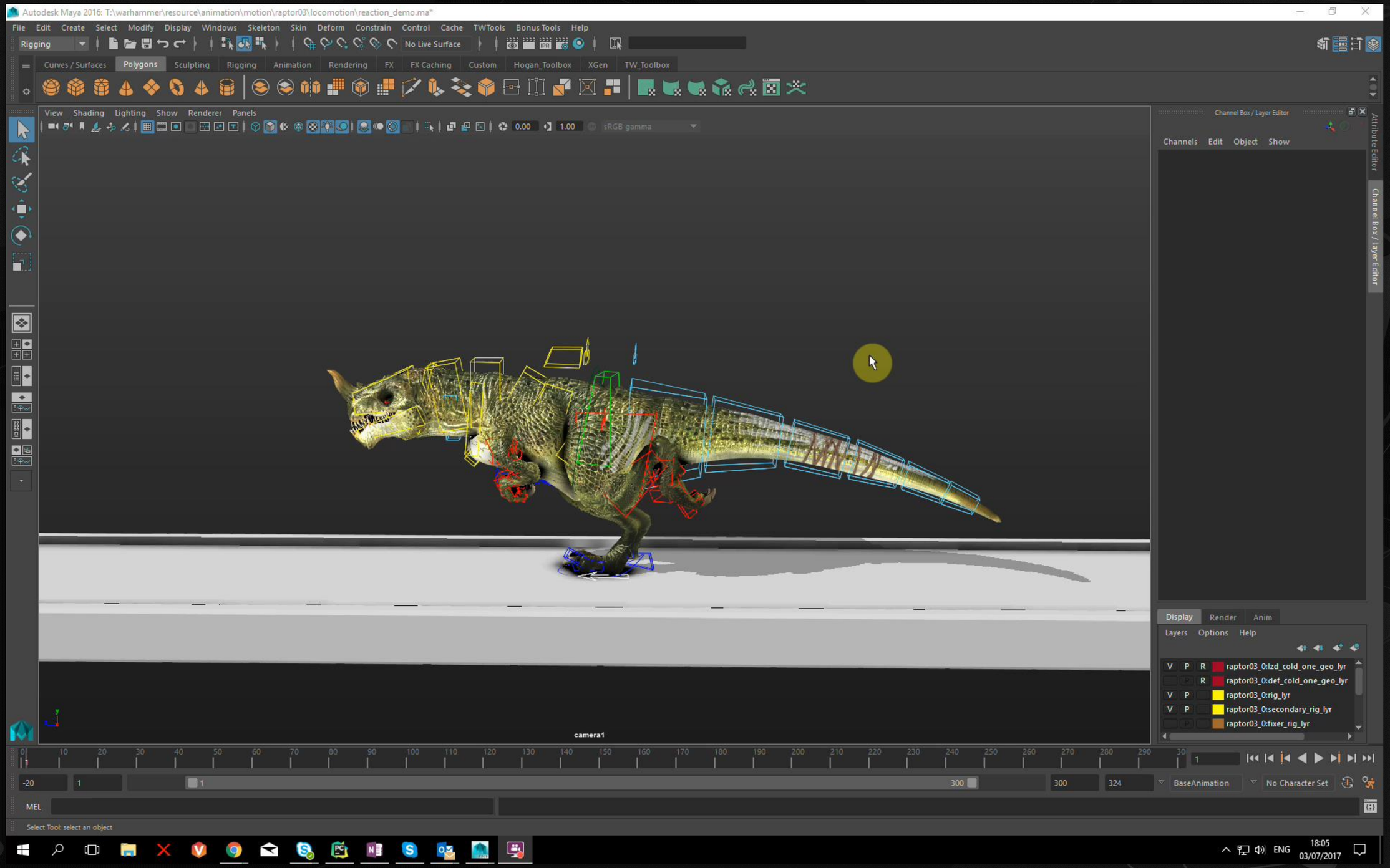
Standard or Reversed Pole Vector

Ability to rebuild consistently (backward compat)

Stretch

Soft Ik

Forward Facing Rigging


Autodesk Maya 2016: T:\warhammer\resource\animation\motion\raptor03\locomotion\reaction_demo.ma*
File Edit Create Select Modify Display Windows Skeleton Skin Deform Constrain Control Cache TWTools Bonus Tools Help
Rigging
No Live Surface
Curves / Surfaces Polygons Sculpting Rigging Animation Rendering FX FX Caching Custom Hogan_Toolbox XGen TW_Toolbox
View Shading Lighting Show Renderer Panels
Channel Box / Layer Editor
Channels Edit Object Show
Attribute Editor
Channel Box / Layer Editor
Display Render Anim
Layers Options Help
raptor03_0:lzd_cold_one_geo_lyr
raptor03_0:def_cold_one_geo_lyr
raptor03_0:rig_lyr
raptor03_0:secondary_rig_lyr
raptor03_0:fixer_rig_lyr
camera1
BaseAnimation
No Character Set
MEL
Select Tool: select an object
ENG
18:05
03/07/2017

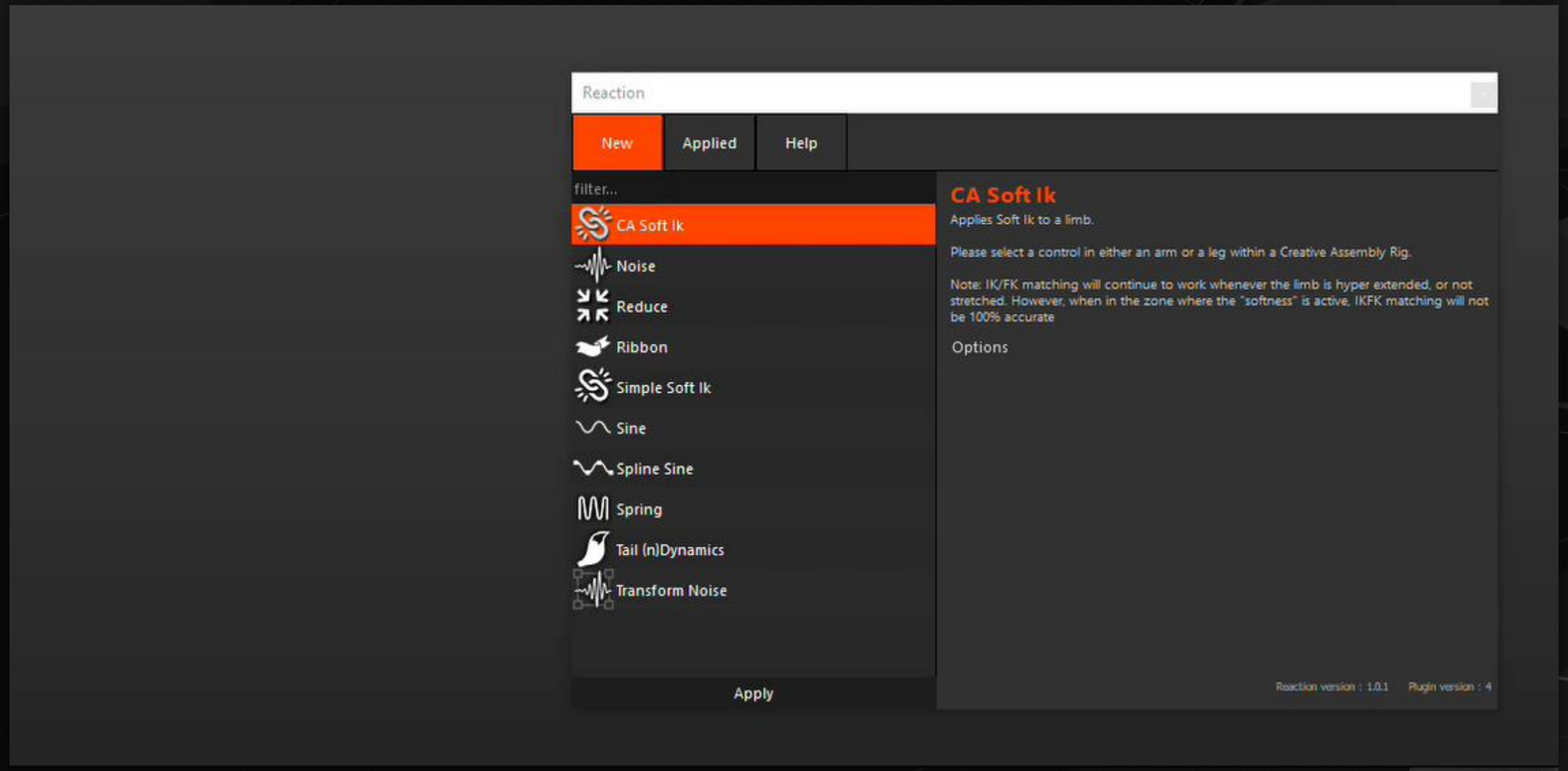
Abstract Methods:

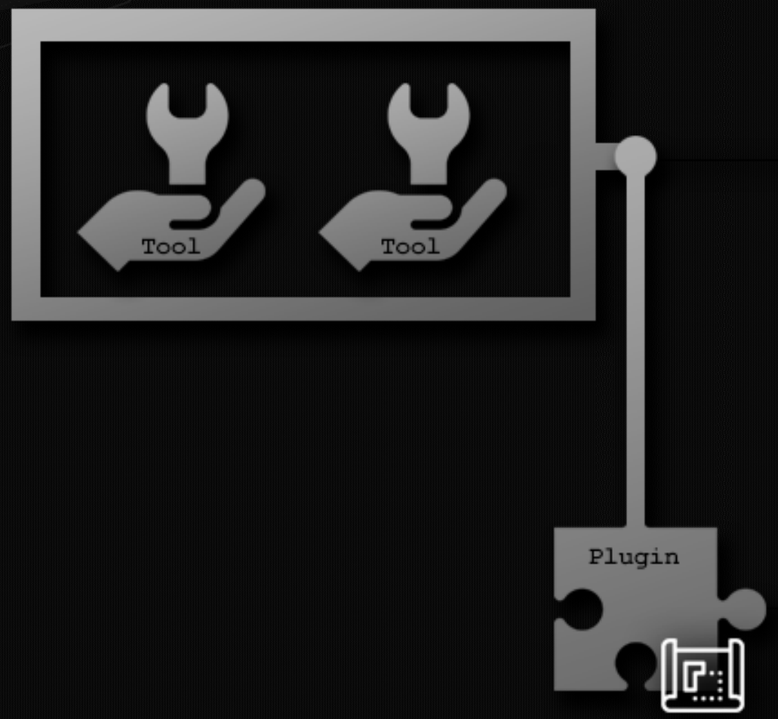
Apply

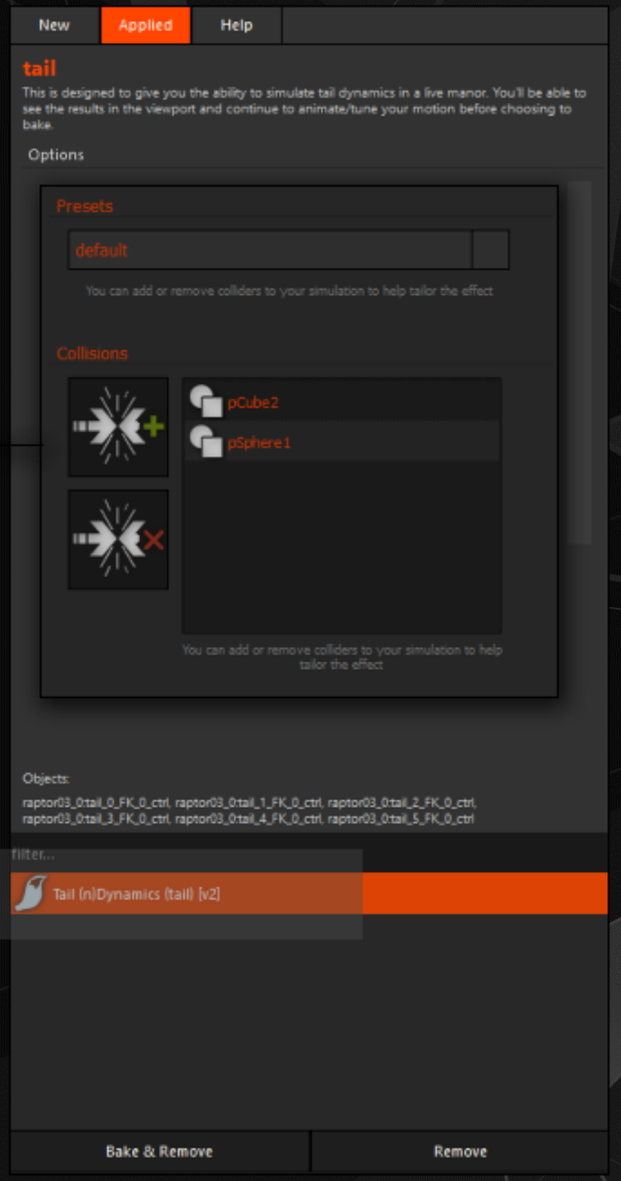
Remove(bake)

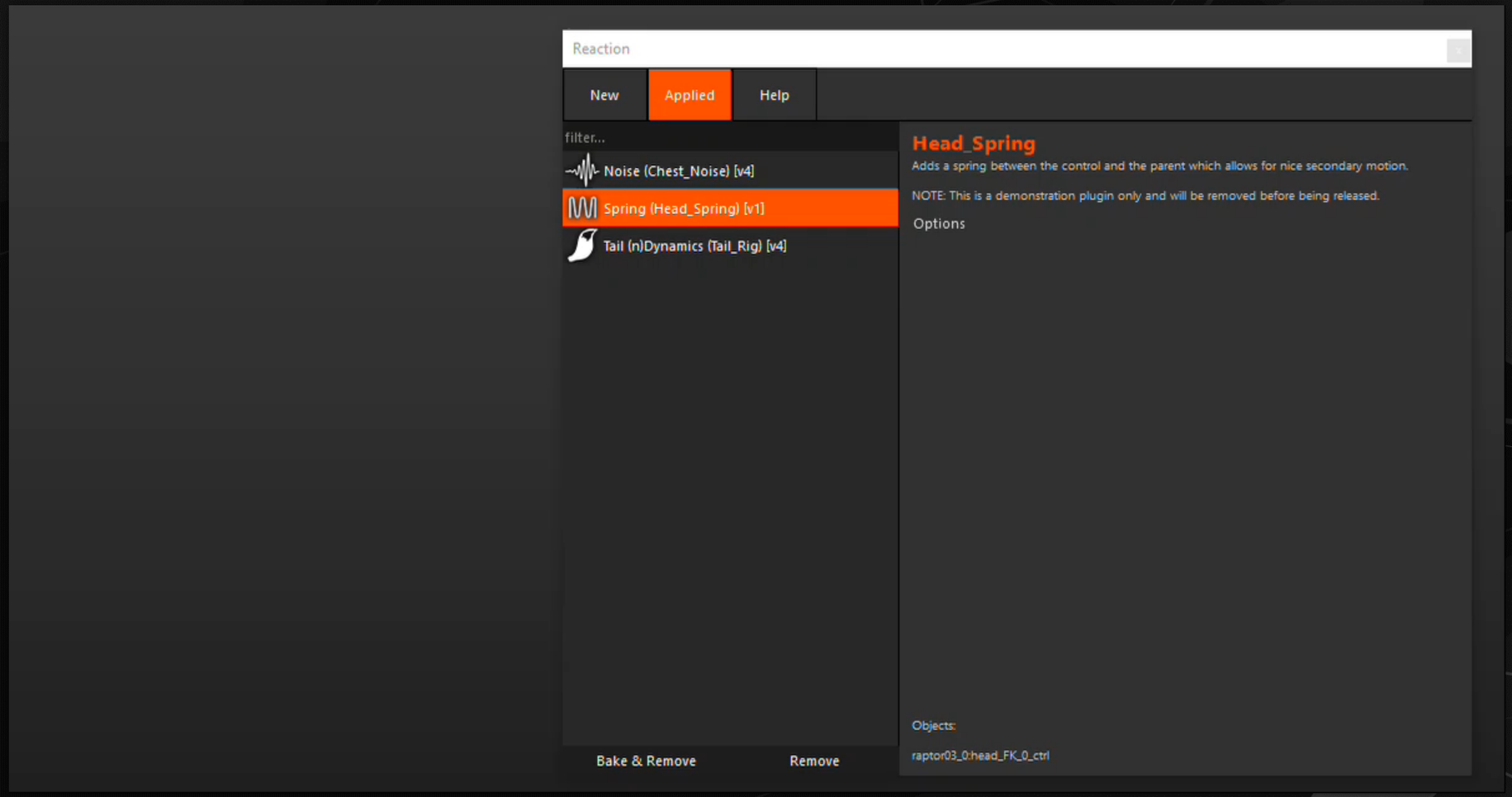
Build Ui

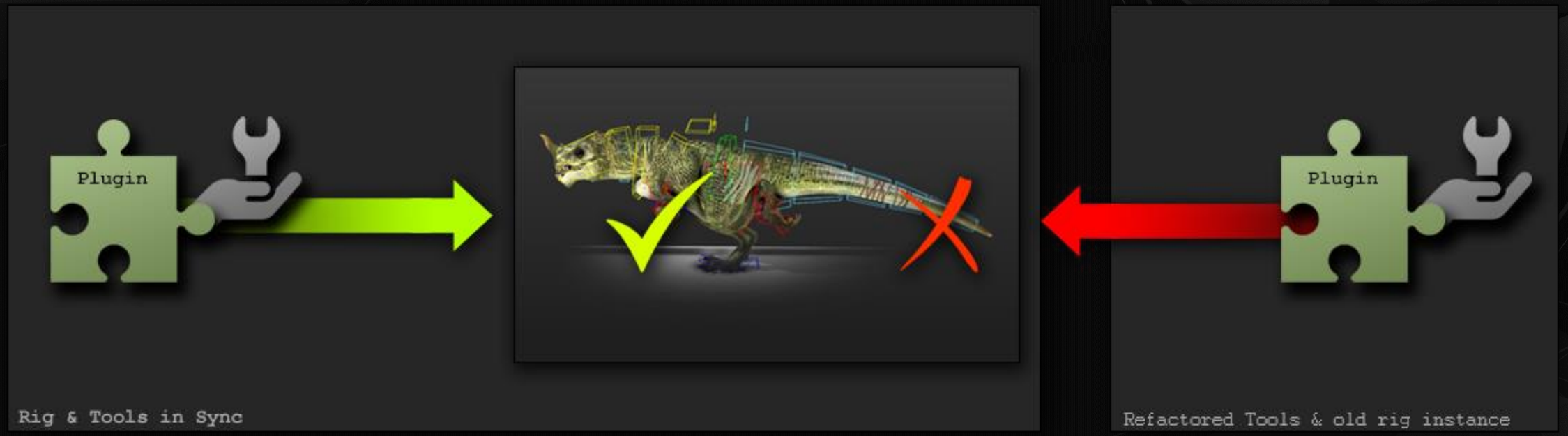
Runtime Ui

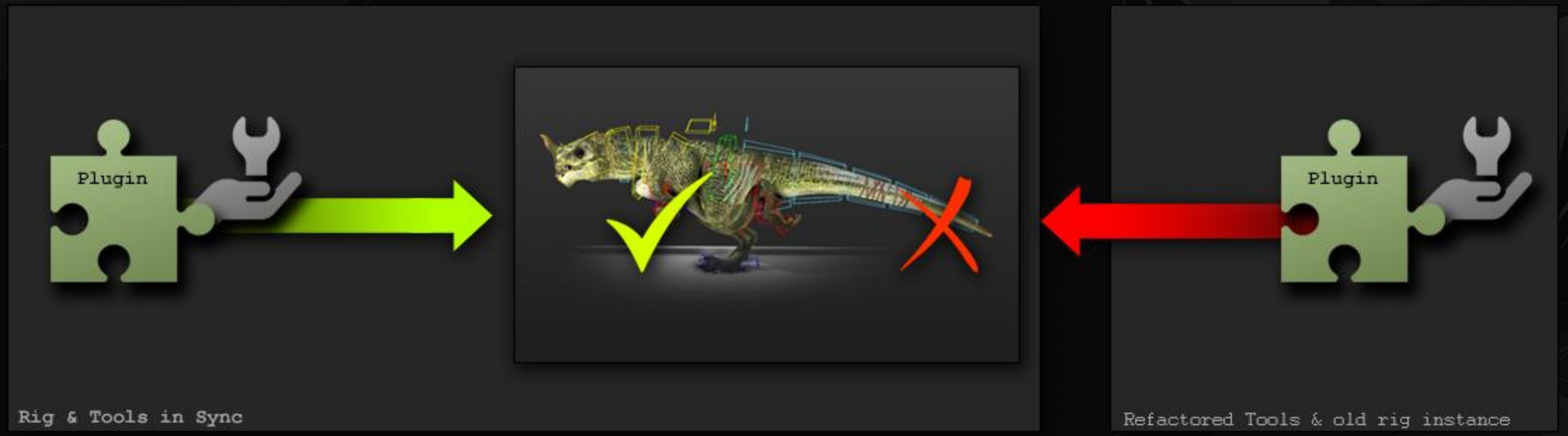

Reaction
New
Applied
Help
filter...
CA Soft Ik
Noise
Reduce
Ribbon
Simple Soft Ik
Sine
Spline Sine
Spring
Tail (n)Dynamics
Transform Noise
Apply
CA Soft Ik
Applies Soft Ik to a limb.
Please select a control in either an arm or a leg within a Creative Assembly Rig.
Note: IK/FK matching will continue to work whenever the limb is hyper extended, or not stretched. However, when in the zone where the "softness" is active, IKFK matching will not be 100% accurate
Options
Reaction version : 1.0.1
Plugin version : 4

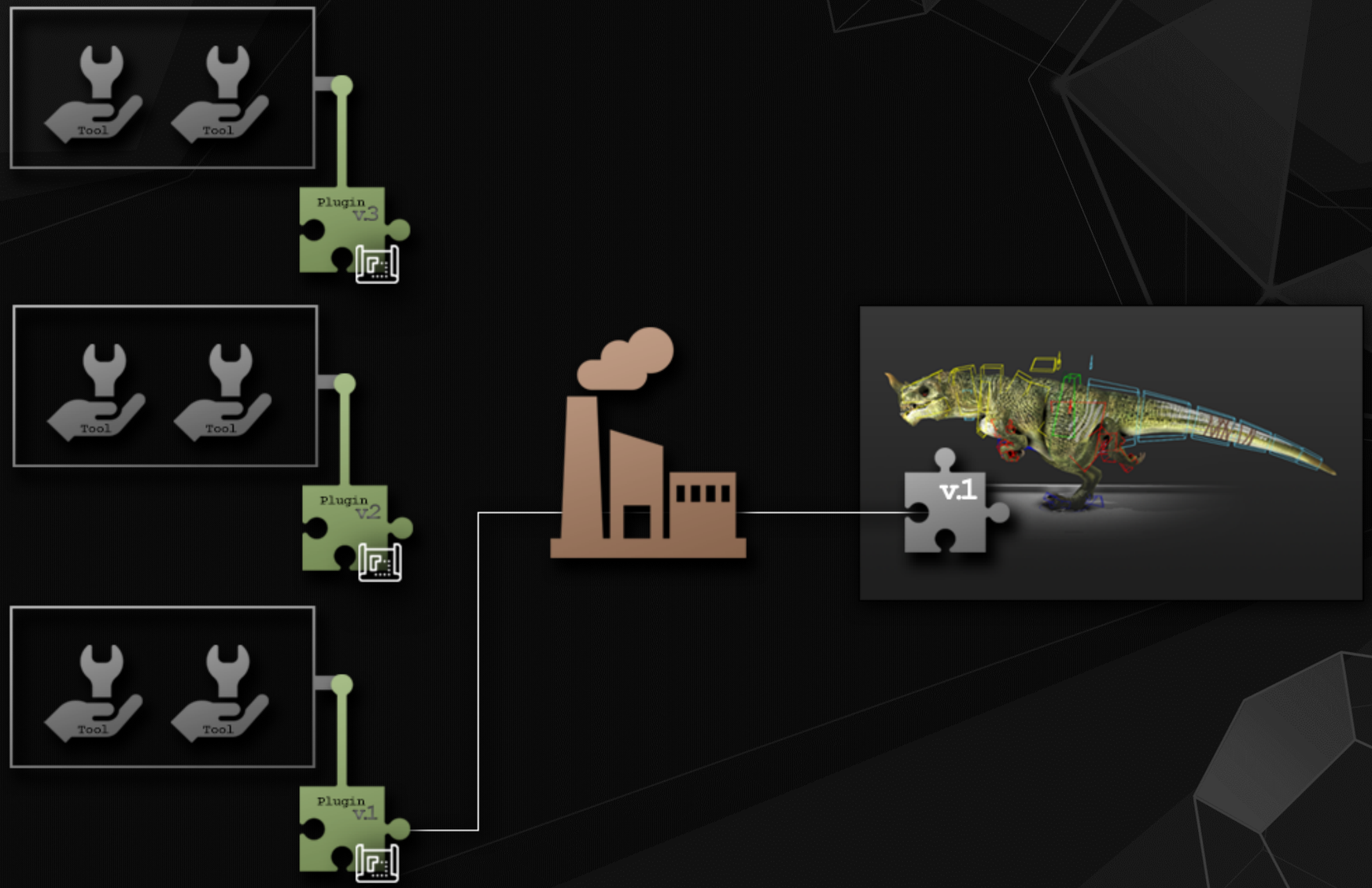

Tool
Tool
Plugin

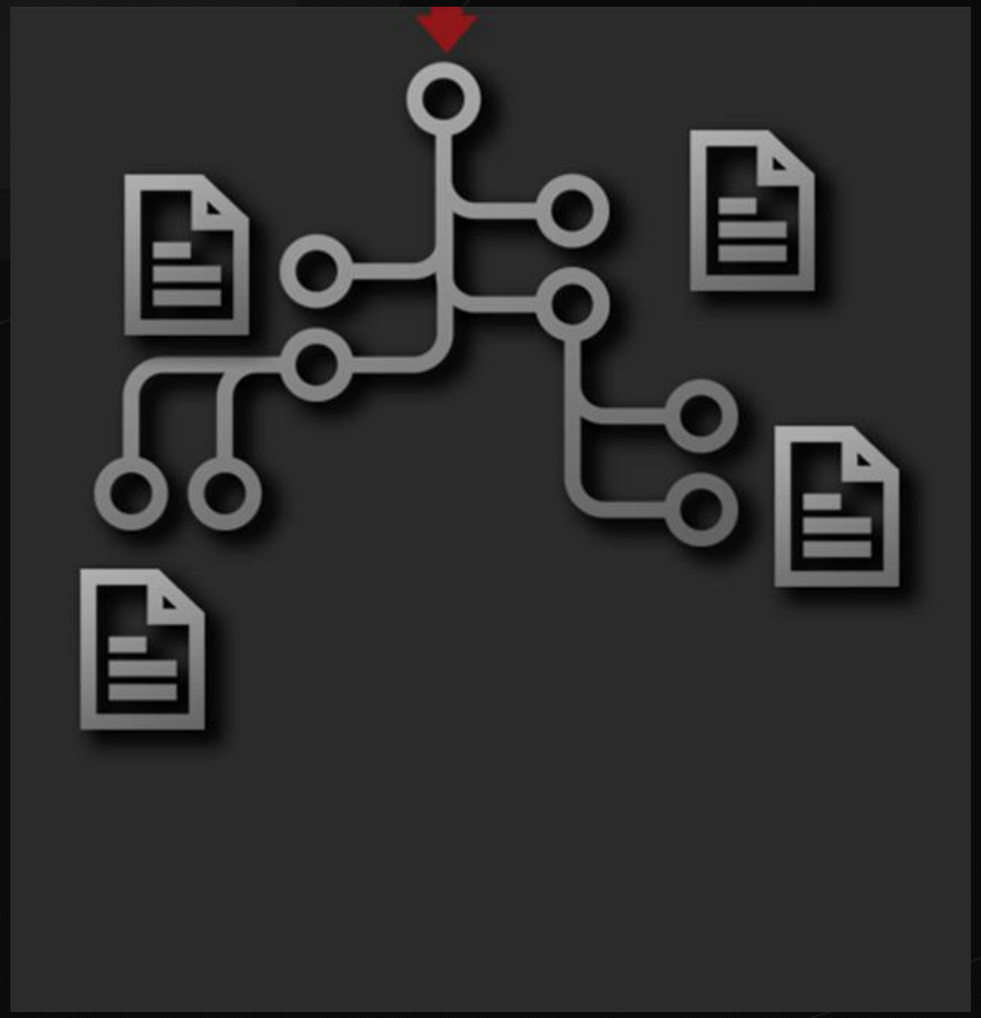

New
Applied
Help
tail
This is designed to give you the ability to simulate tail dynamics in a live manor. You'll be able to see the results in the viewport and continue to animate/tune your motion before choosing to bake.
Options
Presets
default
You can add or remove colliders to your simulation to help tailor the effect
Collisions
pCube2
pSphere1
You can add or remove colliders to your simulation to help tailor the effect
Objects:
raptor03_0tail_0_FK_0_ctrl, raptor03_0tail_1_FK_0_ctrl, raptor03_0tail_2_FK_0_ctrl, raptor03_0tail_3_FK_0_ctrl, raptor03_0tail_4_FK_0_ctrl, raptor03_0tail_5_FK_0_ctrl
filter...
Tail (n)Dynamics (tail) [v2]
Bake & Remove
Remove

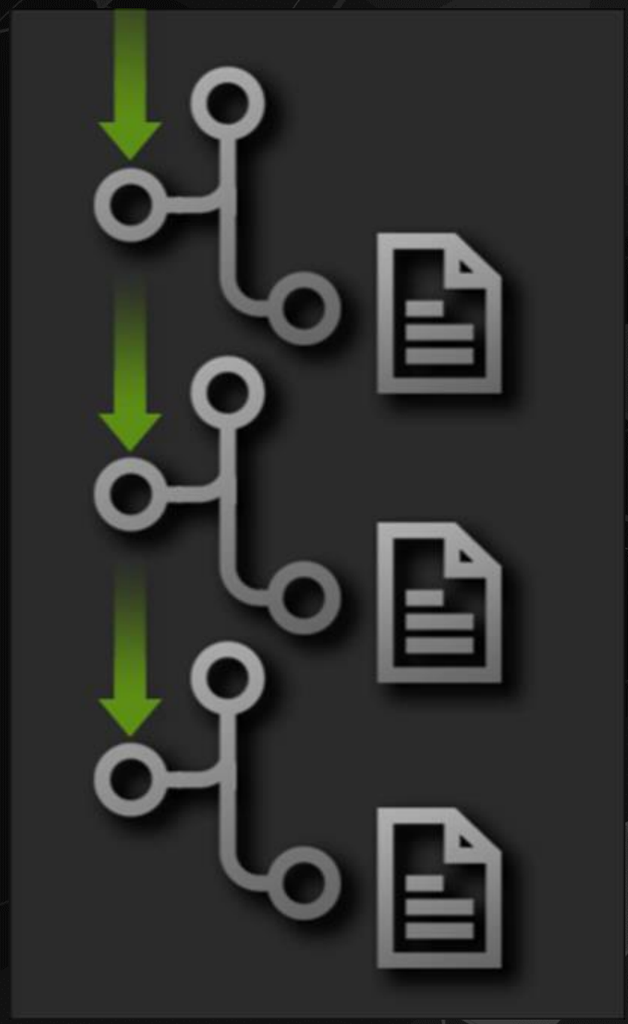

Reaction
New
Applied
Help
filter...
Noise (Chest_Noise) [v4]
Spring (Head_Spring) [v1]
Tail (n)Dynamics (Tail_Rig) [v4]
Bake & Remove
Remove
Head_Spring
Adds a spring between the control and the parent which allows for nice secondary motion.
NOTE: This is a demonstration plugin only and will be removed before being released.
Options
Objects:
raptor03_0:head_FK_0_ctrl

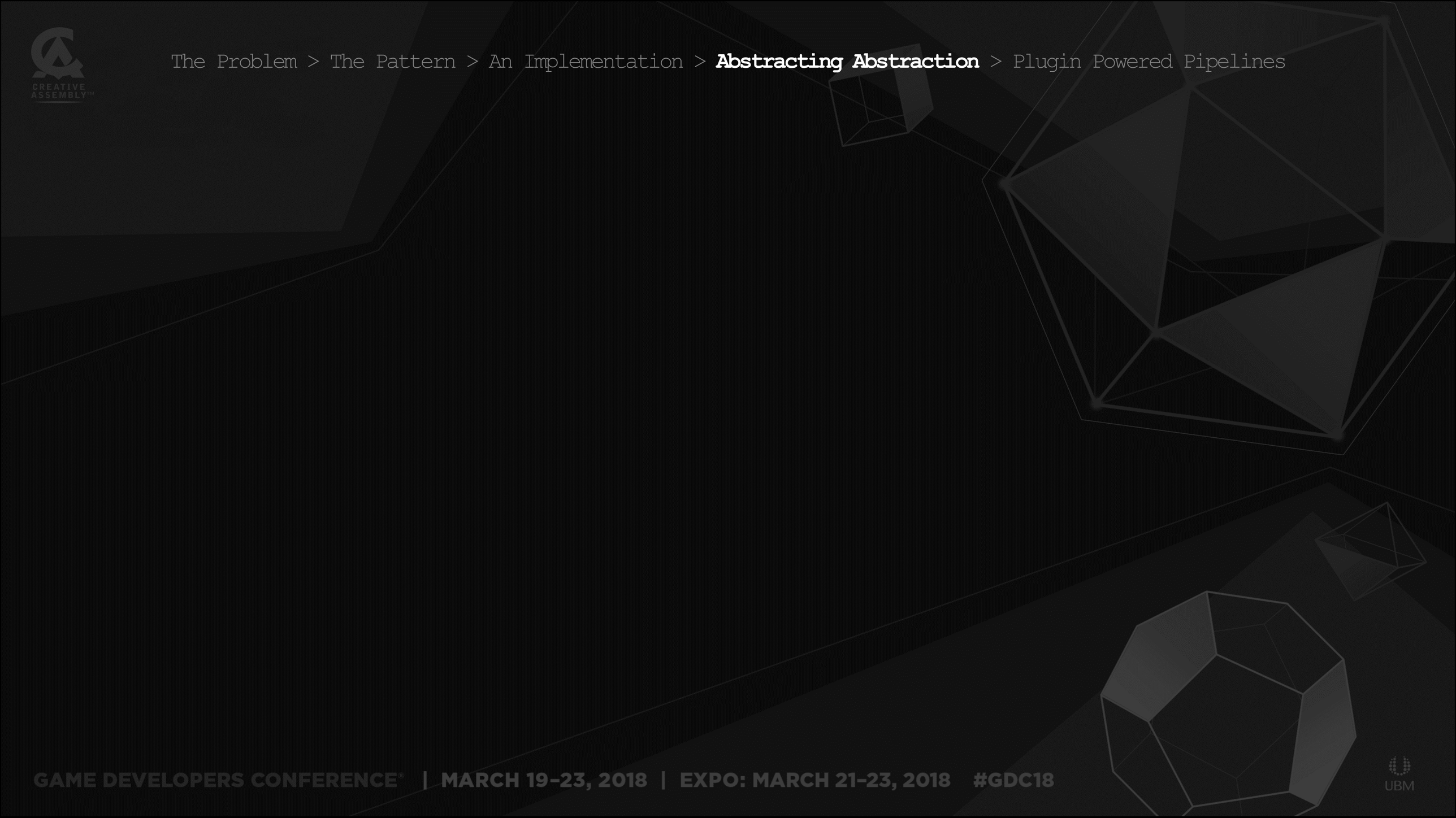

Plugin
Rig & Tools in Sync
Plugin
Refactored Tools & old rig instance


Plugin
Rig & Tools in Sync
Plugin
Refactored Tools & old rig instance


Tool
Tool
Plugin v.3
Tool
Tool
Plugin v.2
Tool
Tool
Plugin v.1
v.1

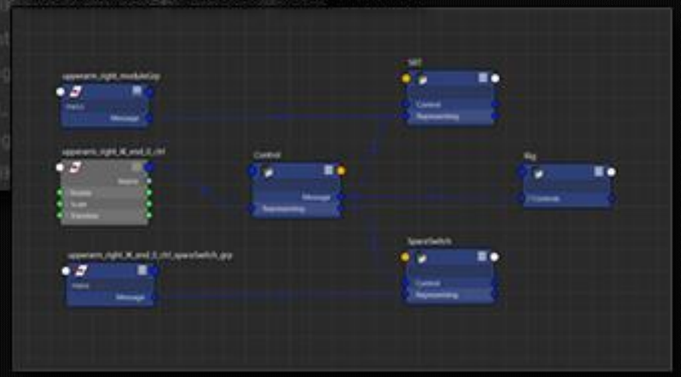




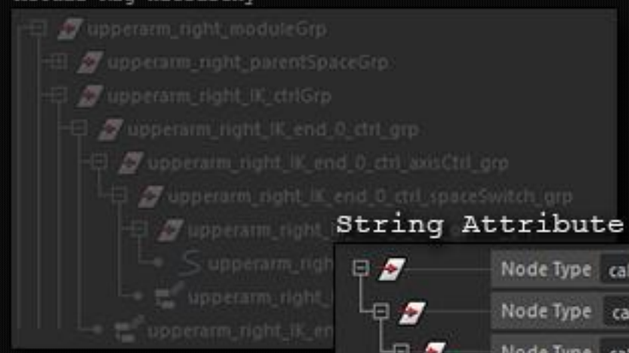


Actual Rig Hierarchy

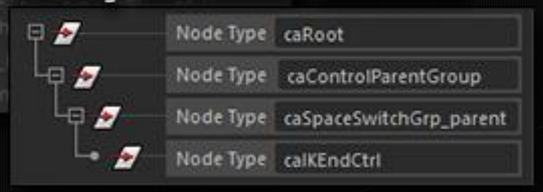
Network Based Metadata



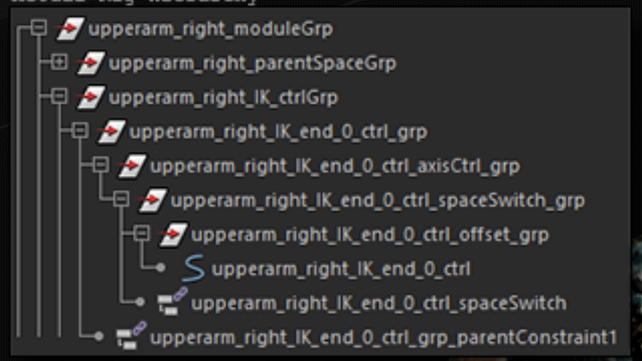
Actual Rig Hierarchy



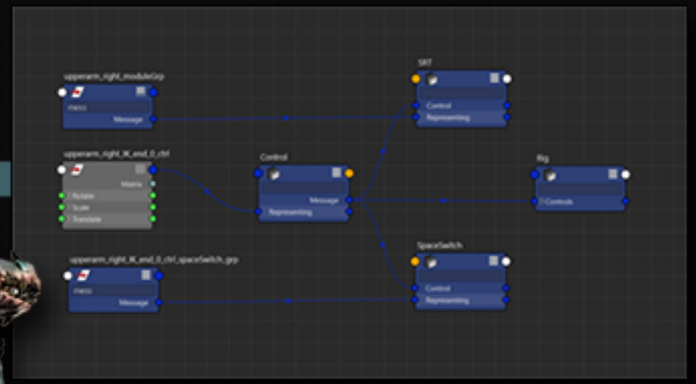
String Attribute Metadata



Actual Rig Hierarchy

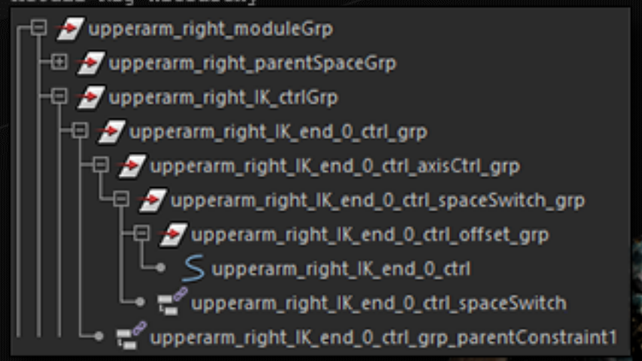


Network Based Metadata

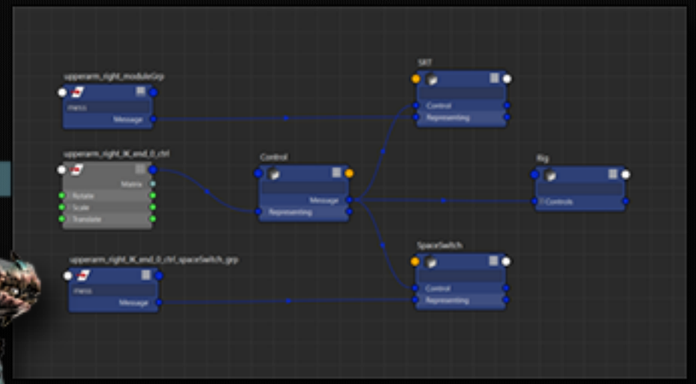


```python
def get_opposing_spaceswitch(control):
    """
    Returns the space switch node for the opposing control

    :param control: pm.nt.DagNode

    :return: pm.nt.DagNode
    """
    # -- As for permission rather than forgiveness...
    try:
        # -- Get the meta node for the selected object
        meta_node = control.message.outputs(
            type='CAMetaControl'
        )[0]

        # -- Look for the opposing (left <-> right)
        opposing_meta = meta_node.opposing.inputs()[0]

        # -- Now get the space switch attribute
        spaceswitch_meta = opposing_meta.message.outputs(
            type='CAMetaSpaceSwitch'
        )[0]

        return spaceswitch_meta

    except :
        return None
```

Actual Rig Hierarchy



Network Based Metadata

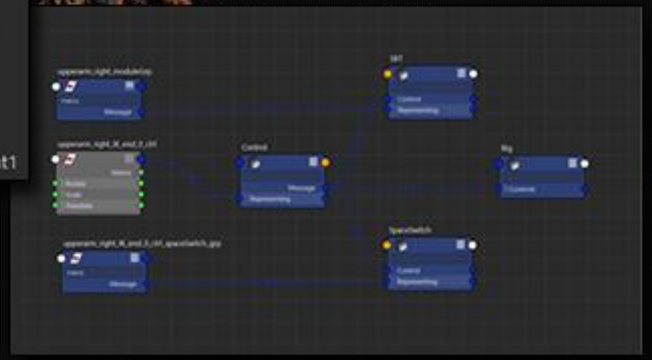


```python
def get_opposing_spaceswitch(control):
    """
    Returns the space switch node for the opposing control

    :param control: pm.nt.DagNode

    :return: pm.nt.DagNode
    """
    # -- As for permission rather than forgiveness...
    try:
        # -- Get the meta node for the selected object
        meta_node = control.message.outputs(
            type='CAMetaControl'
        )[0]

        # -- Look for the opposing (left <-> right)
        opposing_meta = meta_node.opposing.inputs()[0]

        # -- Now get the space switch attribute
        spaceswitch_meta = opposing_meta.message.outputs(
            type='CAMetaSpaceSwitch'
        )[0]

        return spaceswitch_meta

    except :
        return None
```

Actual Rig Hierarchy

- upperarm_right_moduleGrp
  - upperarm_right_parentSpaceGrp
  - upperarm_right_IK_ctrlGrp
    - upperarm_right_IK_end_0_ctrl_grp
      - upperarm_right_IK_end_0_ctrl_axisCtrl_grp
        - upperarm_right_IK_end_0_ctrl_spaceSwitch_grp
          - upperarm_right_IK_end_0_ctrl_offset_grp
            - upperarm_right_IK_end_0_ctrl
            - upperarm_right_IK_end_0_ctrl_spaceSwitch
            - upperarm_right_IK_end_0_ctrl_grp_parentConstraint1

Network Based Metadata



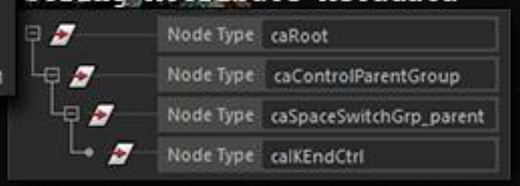
?

```
def get_opposing_spaceswitch(control):
    """
    Returns the space switch node for the opposing control

    :param control: Variable Data Type

    :return: pm.nt.DagNode
    """
    # -- Do some form of cast to get the metadata object
    meta = MetaSystem.get(control)

    # -- Use a simple API to get what you need
    return meta.opposing.spaceswitch.get('pymel')
```

Actual Rig Hierarchy

- upperarm_right_moduleGrp
  - upperarm_right_parentSpaceGrp
  - upperarm_right_IK_ctrlGrp
    - upperarm_right_IK_end_0_ctrl_grp
      - upperarm_right_IK_end_0_ctrl_axisCtrl_grp
        - upperarm_right_IK_end_0_ctrl_spaceSwitch_grp
          - upperarm_right_IK_end_0_ctrl_offset_grp
            - upperarm_right_IK_end_0_ctrl
            - upperarm_right_IK_end_0_ctrl_spaceSwitch
            - upperarm_right_IK_end_0_ctrl_grp_parentConstraint1

String Attribute Metadata



Actual Rig Hierarchy

- upperarm_right_moduleGrp
  - upperarm_right_parentSpaceGrp
  - upperarm_right_IK_ctrlGrp
    - upperarm_right_IK_end_0_ctrl_grp
      - upperarm_right_IK_end_0_ctrl_axisCtrl_grp
        - upperarm_right_IK_end_0_ctrl_spaceSwitch_grp
          - upperarm_right_IK_end_0_ctrl_offset_grp
            - upperarm_right_IK_end_0_ctrl
            - upperarm_right_IK_end_0_ctrl_spaceSwitch
            - upperarm_right_IK_end_0_ctrl_grp_parentConstraint1

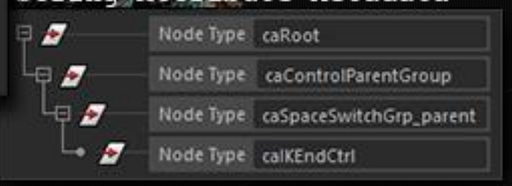
Network Based Metadata

```
def get_opposing_spaceswitch(control):
    """
    Returns the space switch node for the opposing control

    :param control: Variable Data Type

    :return: pm.nt.DagNode
    """
    # -- Do some form of cast to get the metadata object
    meta = MetaSystem.get(control)

    # -- Use a simple API to get what you need
    return meta.opposing.spaceswitch.get('pixel')
```

Actual Rig Hierarchy

- upperarm_right_moduleGrp
  - upperarm_right_parentSpaceGrp
  - upperarm_right_IK_ctrlGrp
    - upperarm_right_IK_end_0_ctrl_grp
      - upperarm_right_IK_end_0_ctrl_axisCtrl_grp
        - upperarm_right_IK_end_0_ctrl_spaceSwitch_grp
          - upperarm_right_IK_end_0_ctrl_offset_grp
            - upperarm_right_IK_end_0_ctrl
            - upperarm_right_IK_end_0_ctrl_spaceSwitch
            - upperarm_right_IK_end_0_ctrl_grp_parentConstraint1

String Attribute Metadata

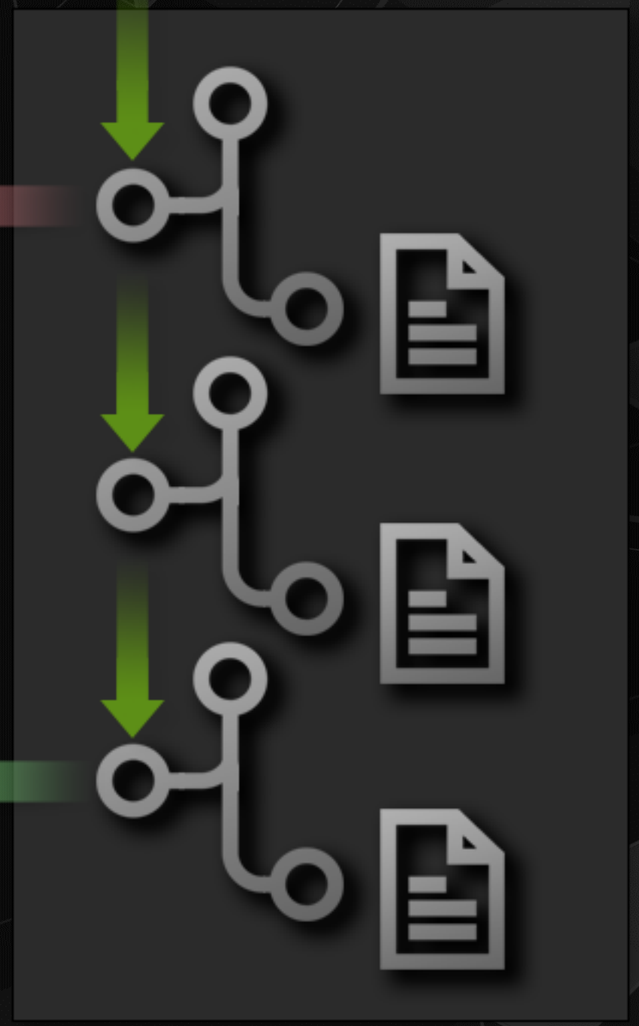


```python
class GripObjectBase(_GripBase):
    __metaclass__ = abc.ABCMeta

    @abc.abstractmethod
    def get(self, api=API_TYPES.GRIP):
        return None

    @abc.abstractmethod
    def tag(self, flag, value):
        return None

    @abc.abstractmethod
    def untag(self, flag, node):
        return None

    @abc.abstractmethod
    def name(self):
        return None

    @abc.abstractmethod
    def state(self):
        return StateFlags.Unknown

    @abc.abstractmethod
    def states(self):
        return StateFlags.Unknown

    @abc.abstractmethod
    def query(self, flags, api=API_TYPES.GRIP):
        return list()

    @abc.abstractmethod
    def children(self, flags, recursive=False, api=API_TYPES.GRIP):
        return list()

    @abc.abstractmethod
    def parents(self, flags, recursive=False, api=API_TYPES.GRIP):
        return list()

    @abc.abstractmethod
    def connected(self, flags, api=API_TYPES.GRIP):
        return list()

    @abc.abstractmethod
    def rig(self):
        return None
```

```
@classmethod
@abc.abstractmethod
def viable(cls, candidate):
    return False
```

```python
class StringMetaPlugin(GripObjectBase):

    @classmethod
    def viable(cls, candidate):
        if maya.cmds.hasAttr(candidate, 'ca_node_type'):
            return True
        return False
```

```python
class MetaNodePlugin(GripObjectBase):

    @classmethod
    def viable(cls, candidate):
        for node in candidate.getParent(-1).message.outputs():
            if isinstance(node, CAMetaRig):
                return True
        return False
```

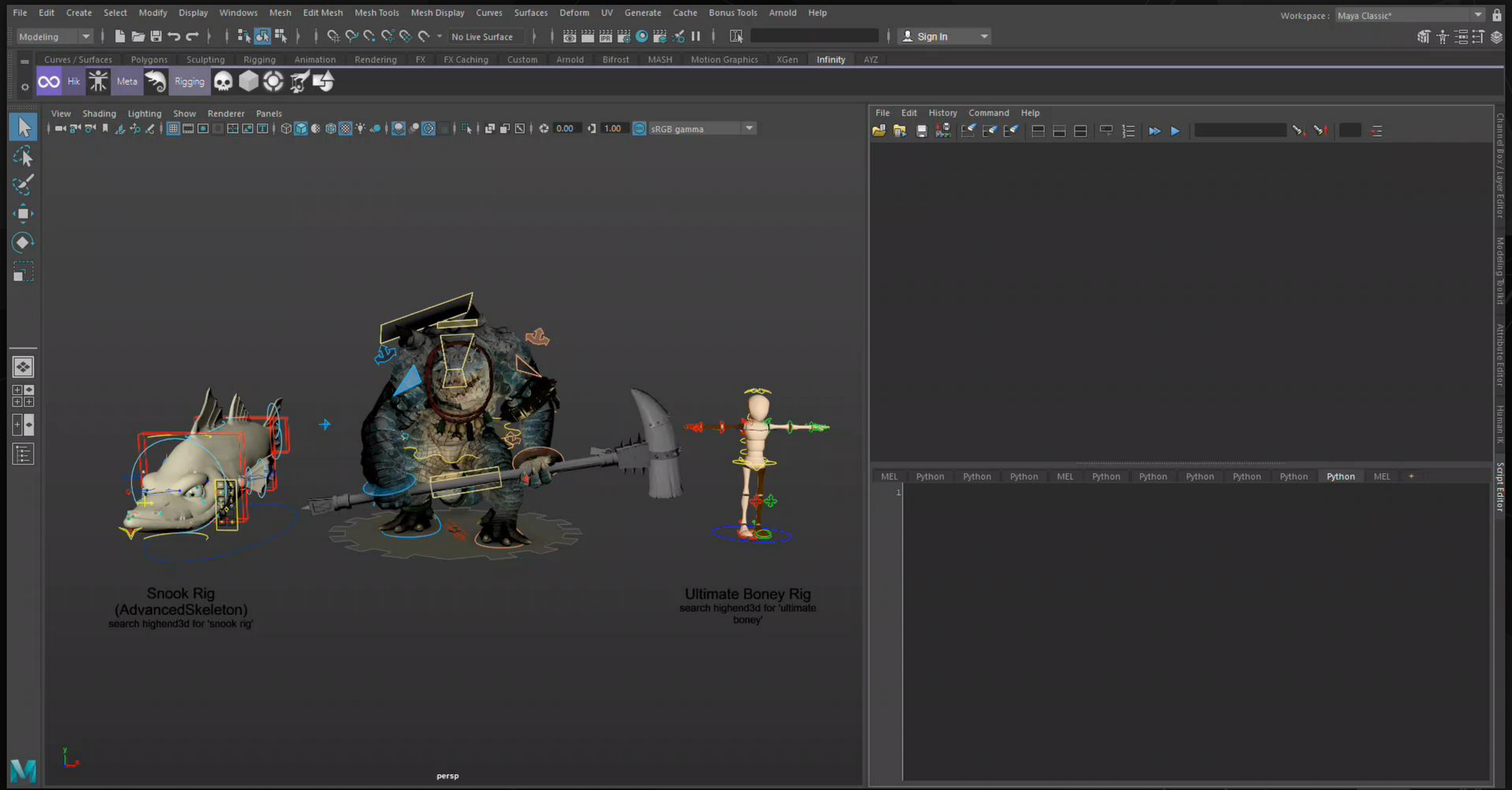


Helper Functionality

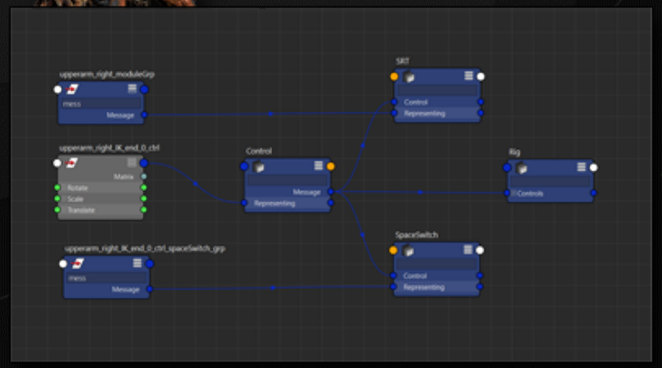
```python
def grab(self, candidate):
    for plugin in self.factory.plugins():
        if plugin.viable(candidate):
            return plugin(candidate)

    raise Exception(
        'No plugin representation for object %s' % candidate
    )
```

Tool Code

```python
# -- Get a metadata representation of the selected
# -- object
meta = grip.grab(pm.selected()[0])

# -- Cycle over all the left controls in the rig
for control in meta.rig().query(RigFlags.Control | RigFlags.Left):

    # -- Get the right control from the left
    opposing_control = control.query(RigFlags.Opposing)

    # -- Get the pymel object, rather than the meta
    # -- object
    node = opposing_control.get('pm')
```

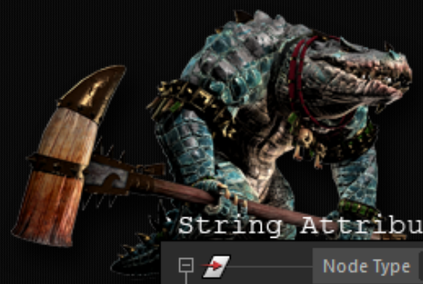

Snook Rig
(AdvancedSkeleton)
search highend3d for 'snook rig'
Ultimate Boney Rig
search highend3d for 'ultimate boney'
persp

Network Based Metadata

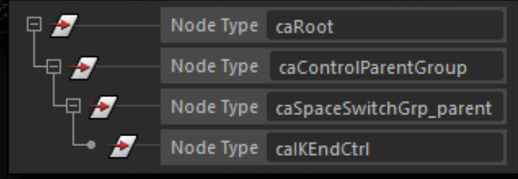


```
class MetaNodePlugin(GripObjectBase):

    def query(self, flags, api=API_TYPES.GRIP):

        if RigFlags.Control in flags:
            return [
                MetaNodePlugin(node)
                for node in self.rig.findChildren(
                    type='CAMetaNodeControl',
                ),
            ]

        return list()
```

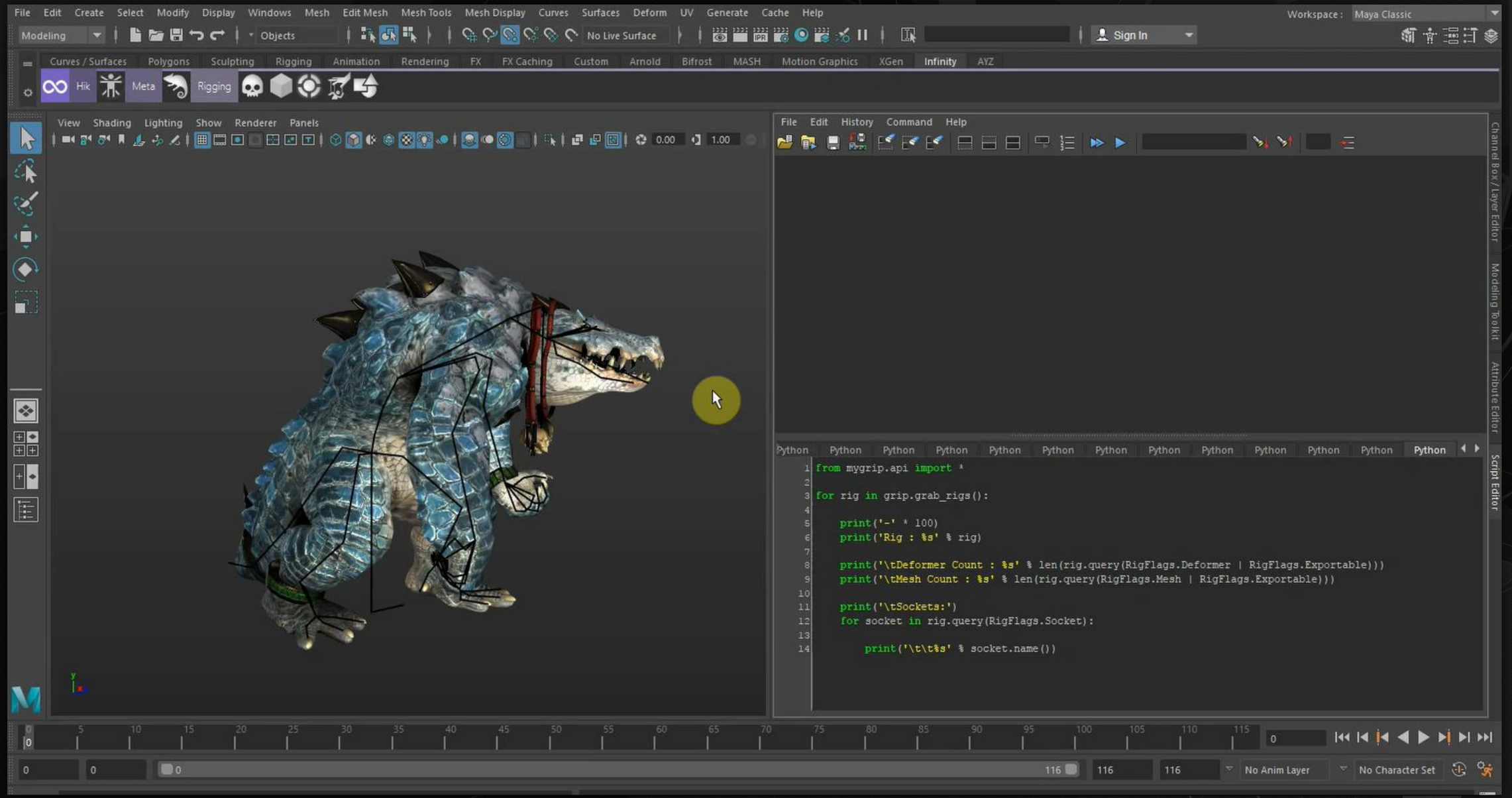
```
>>>
```

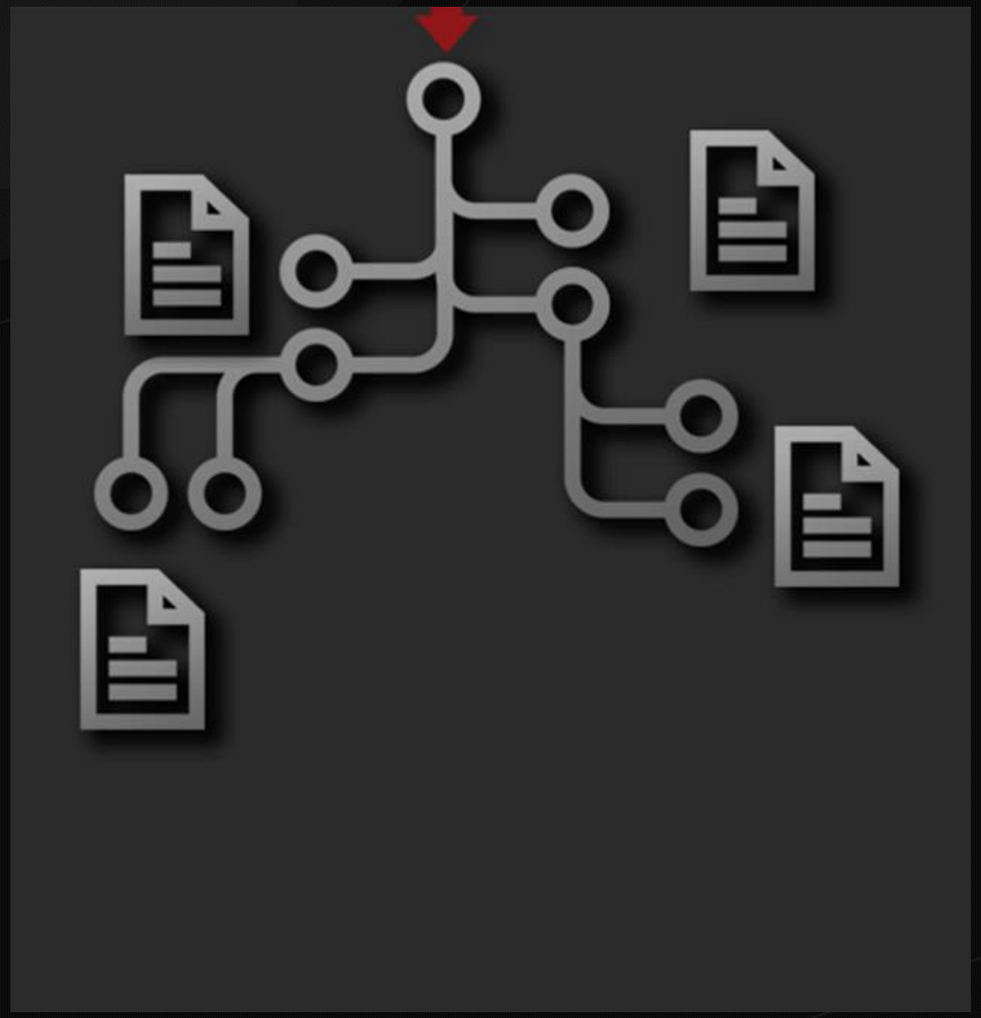


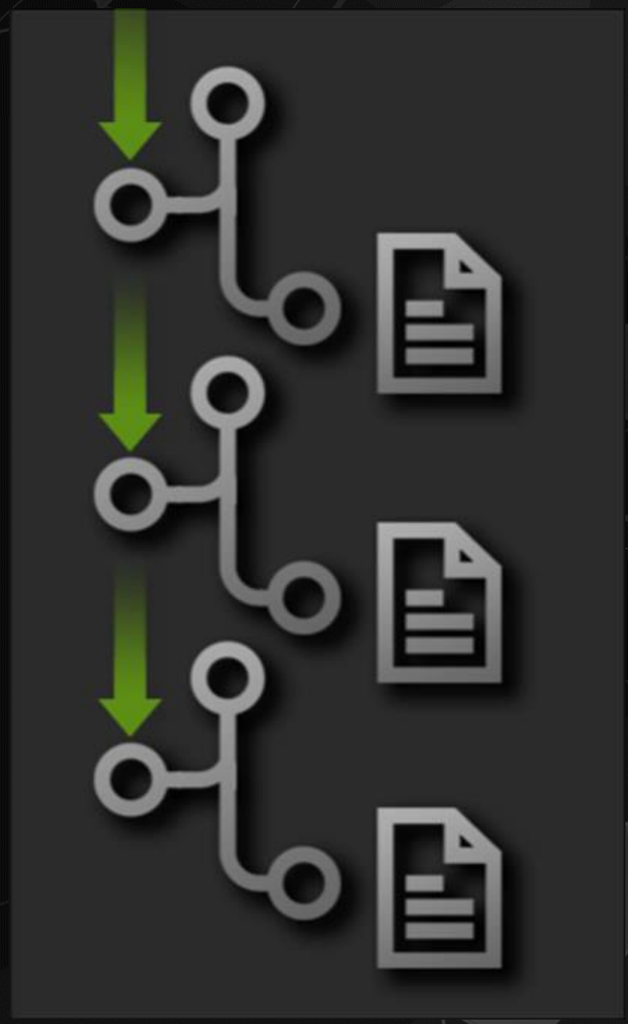
String Attribute Metadata

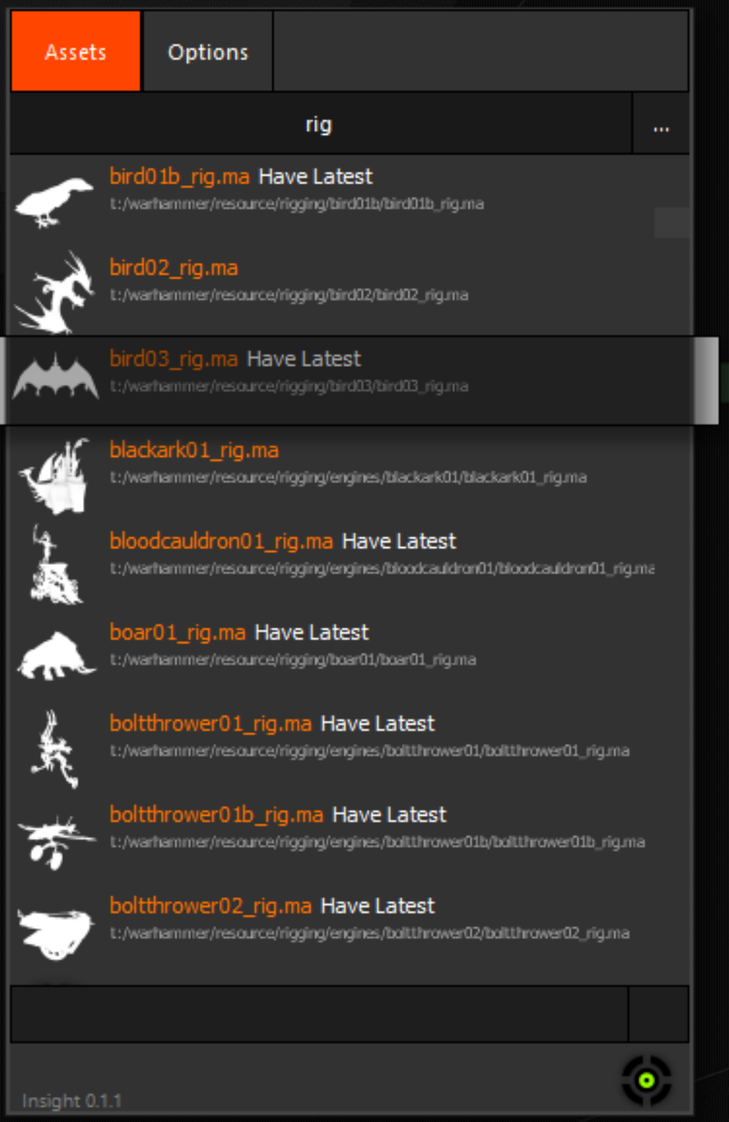



```
class StringMetaPlugin(GripObjectBase):

    def query(self, flags, api=API_TYPES.GRIP):

        if RigFlags.Control in flags:
            meta_attrs = cmds.ls(
                '*.ca_node_type',
                recursive=True,
            )

            for attr in meta_attrs:
                if maya.cmds.getAttr(attr) == self.ControlType:
                    yield MetaNodePlugin(
                        maya.cmds.attributeQuery(attr, node=True)
                    )
```

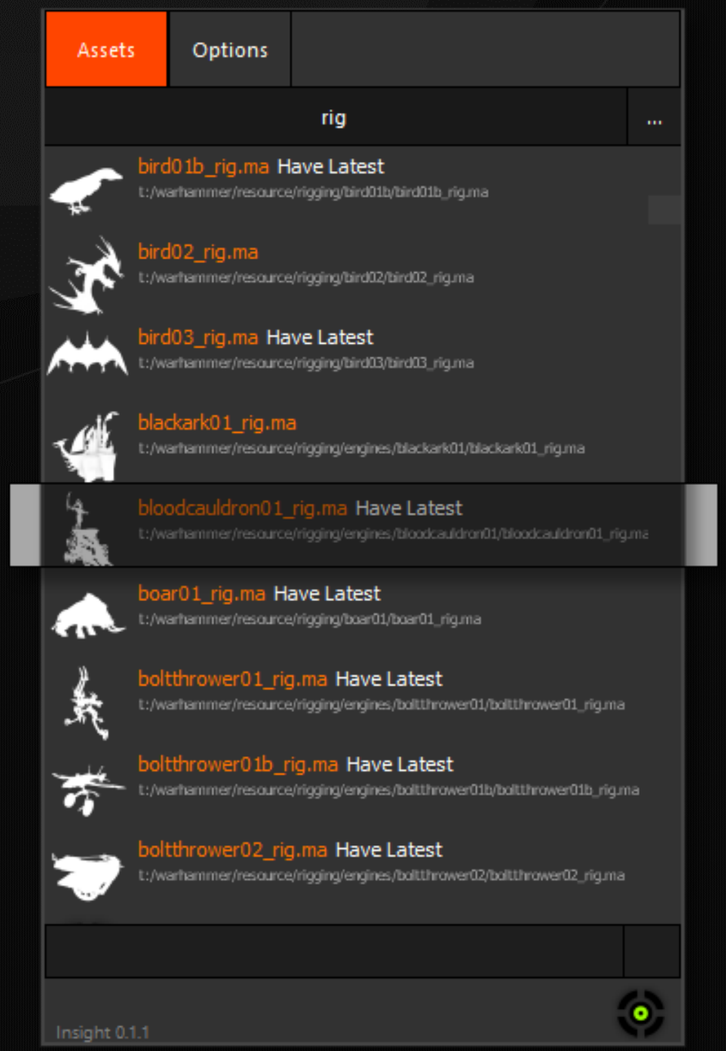

File Edit Create Select Modify Display Windows Mesh Edit Mesh Mesh Tools Mesh Display Curves Surfaces Deform UV Generate Cache Help
Workspace : Maya Classic
Modeling
Objects
No Live Surface
Sign In
Curves / Surfaces Polygons Sculpting Rigging Animation Rendering FX FX Caching Custom Arnold Bifrost MASH Motion Graphics XGen Infinity AYZ
Hik Meta Rigging
View Shading Lighting Show Renderer Panels
File Edit History Command Help
Python
from mygrip.api import *
for rig in grip.grab_rigs():
print('-' * 100)
print('Rig : %s' % rig)
print('\tDeformer Count : %s' % len(rig.query(RigFlags.Deformer | RigFlags.Exportable)))
print('\tMesh Count : %s' % len(rig.query(RigFlags.Mesh | RigFlags.Exportable)))
print('\tSockets:')
for socket in rig.query(RigFlags.Socket):
print('\t\t%s' % socket.name())
No Anim Layer
No Character Set
Channel Box / Layer Editor
Modeling Toolkit
Attribute Editor
Script Editor







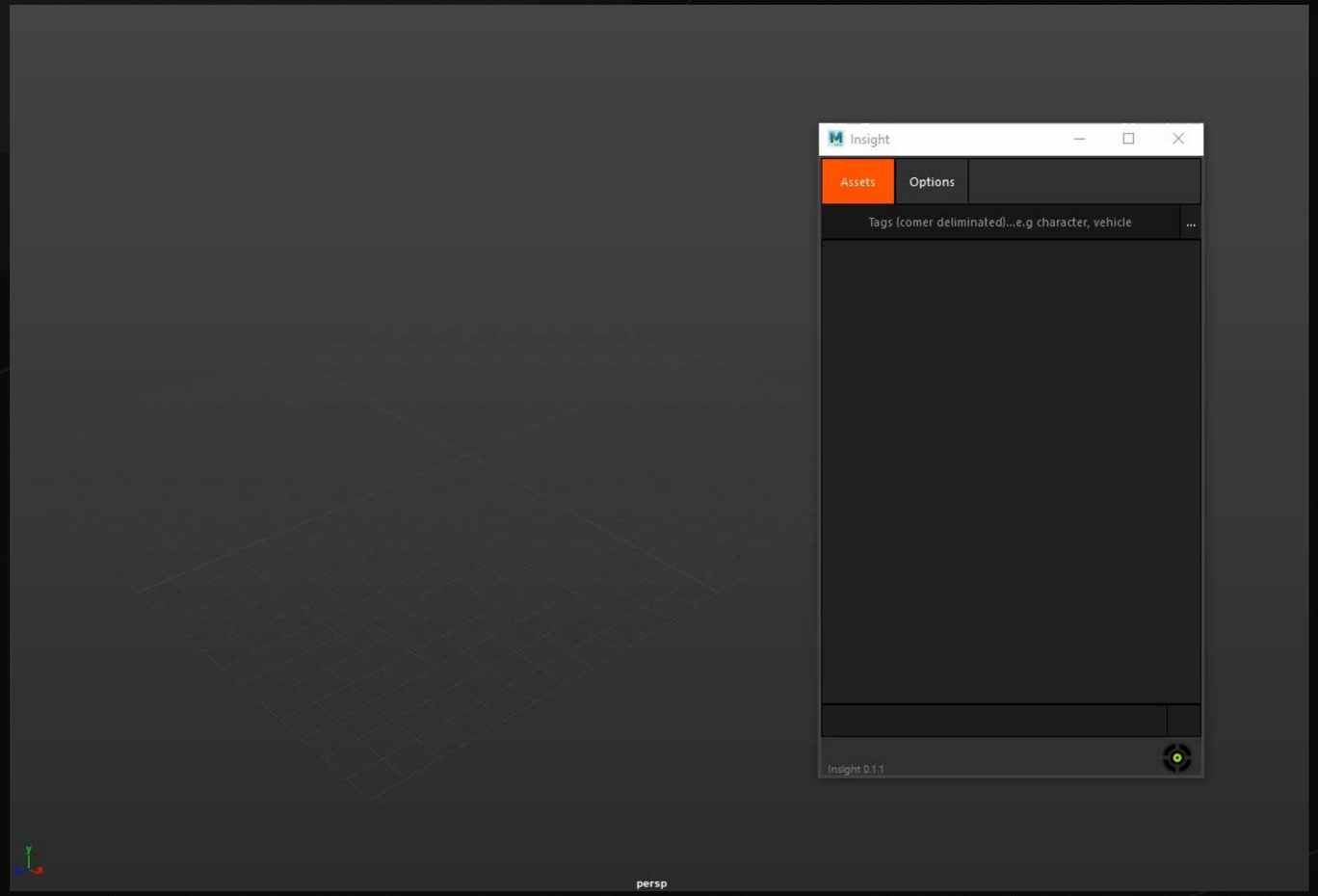

```python
class InsightAssetPlugin(InsightPluginBa
    __metaclass__ = abc.ABCMeta

    requirement = re.compile('', re.I)

    @abc.abstractmethod
    def dependencies(self):
        return list()

    @abc.abstractmethod
    def metadata(self):
        return dict()

    @abc.abstractmethod
    def hash(self):
        return 0

    @abc.abstractmethod
    def auto_generated_tags(self):
        return list()

    @abc.abstractmethod
    def thumbnail(self):
        return None

    @abc.abstractmethod
    def context(self):
        return dict()

    @classmethod
    def viable(cls, filepath):
        return False
```

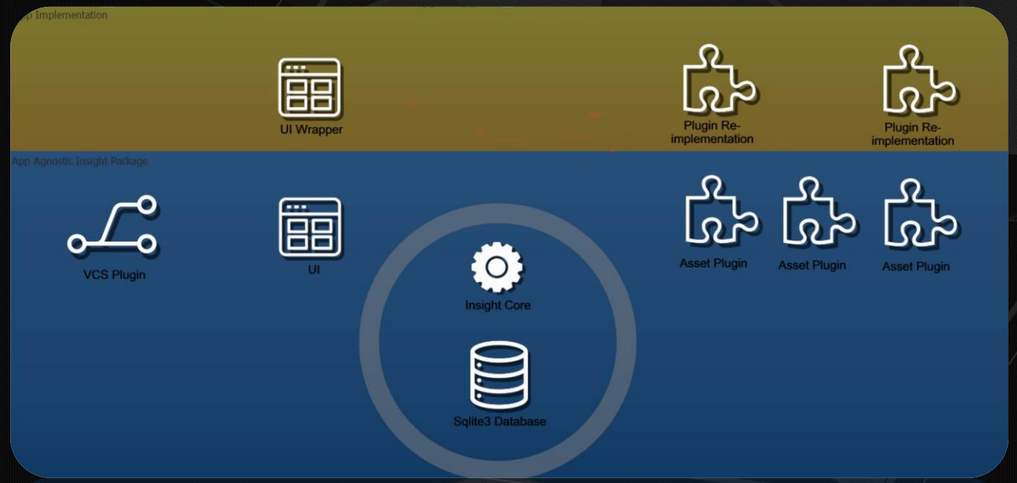


```python
class AnimationAssetPlugin(InsightPluginBase):

    requirement = re.compile('.*(animations).*(\.ma$)', re.I)

    @classmethod
    def viable(cls, filepath):
        return cls.requirement.search(filepath)
```

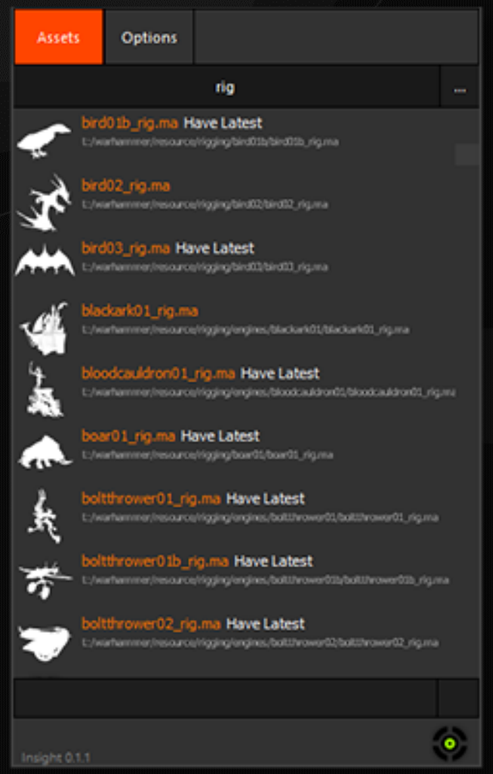
```python
class RigAssetPlugin(InsightPluginBase):

    requirement = re.compile('.*(rigging).*(_rig\.ma$)', re.I)

    @classmethod
    def viable(cls, filepath):
        return cls.requirement.search(filepath)
```

```python
class MeshAssetPlugin(InsightPluginBase):

    requirement = re.compile('.*(art).*(\.max$)', re.I)

    @classmethod
    def viable(cls, filepath):
        return cls.requirement.search(filepath)
```

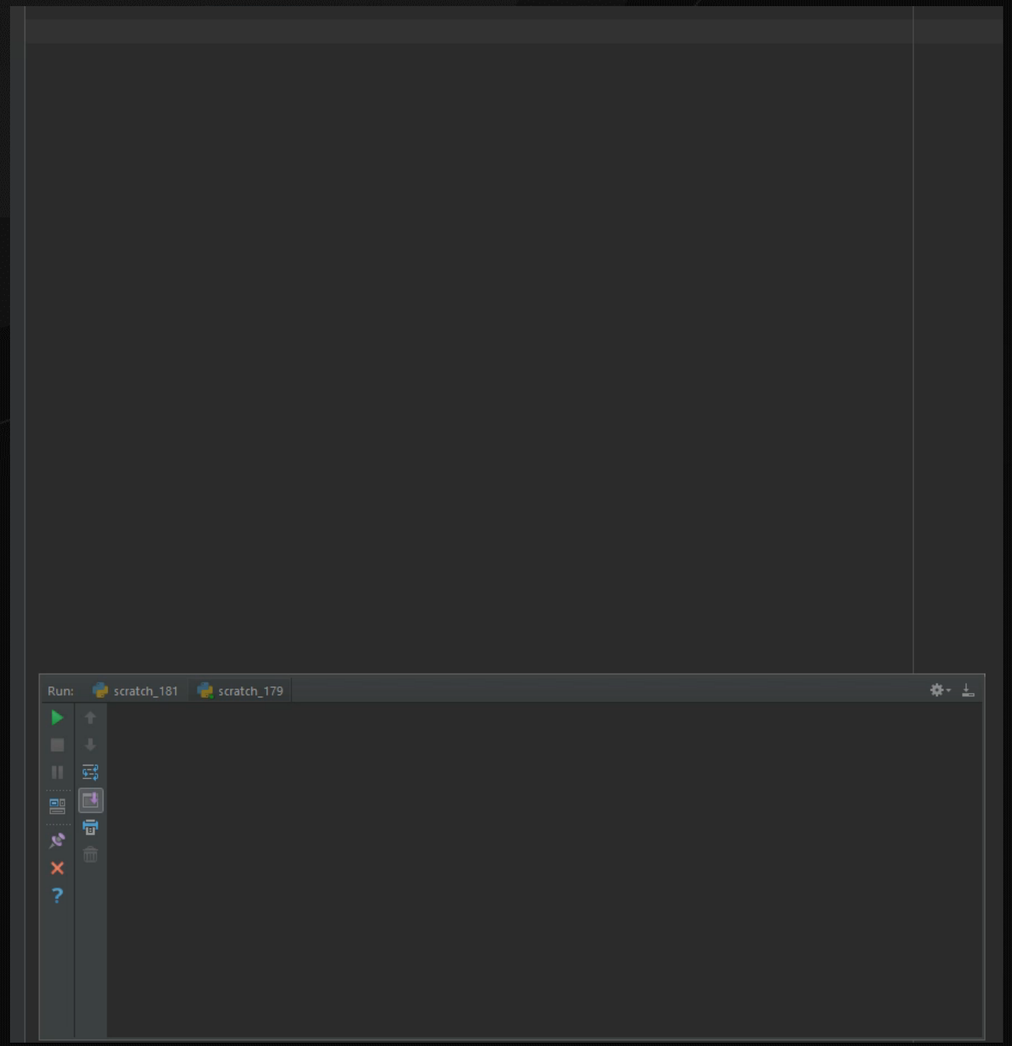


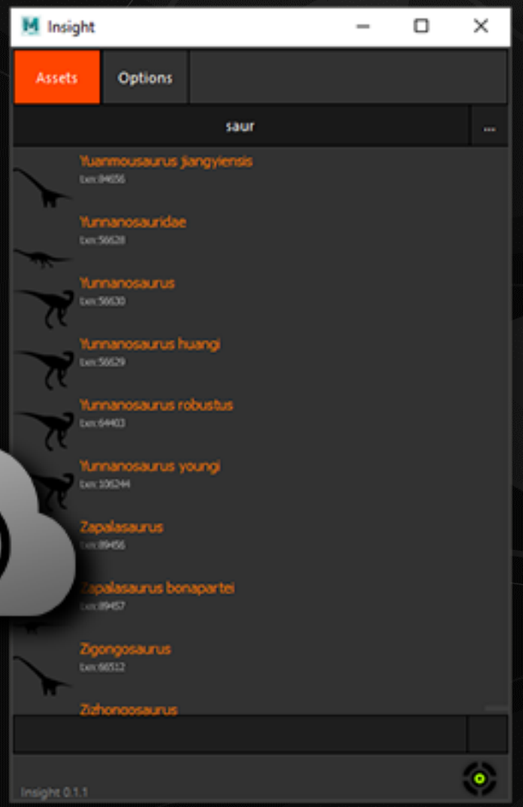


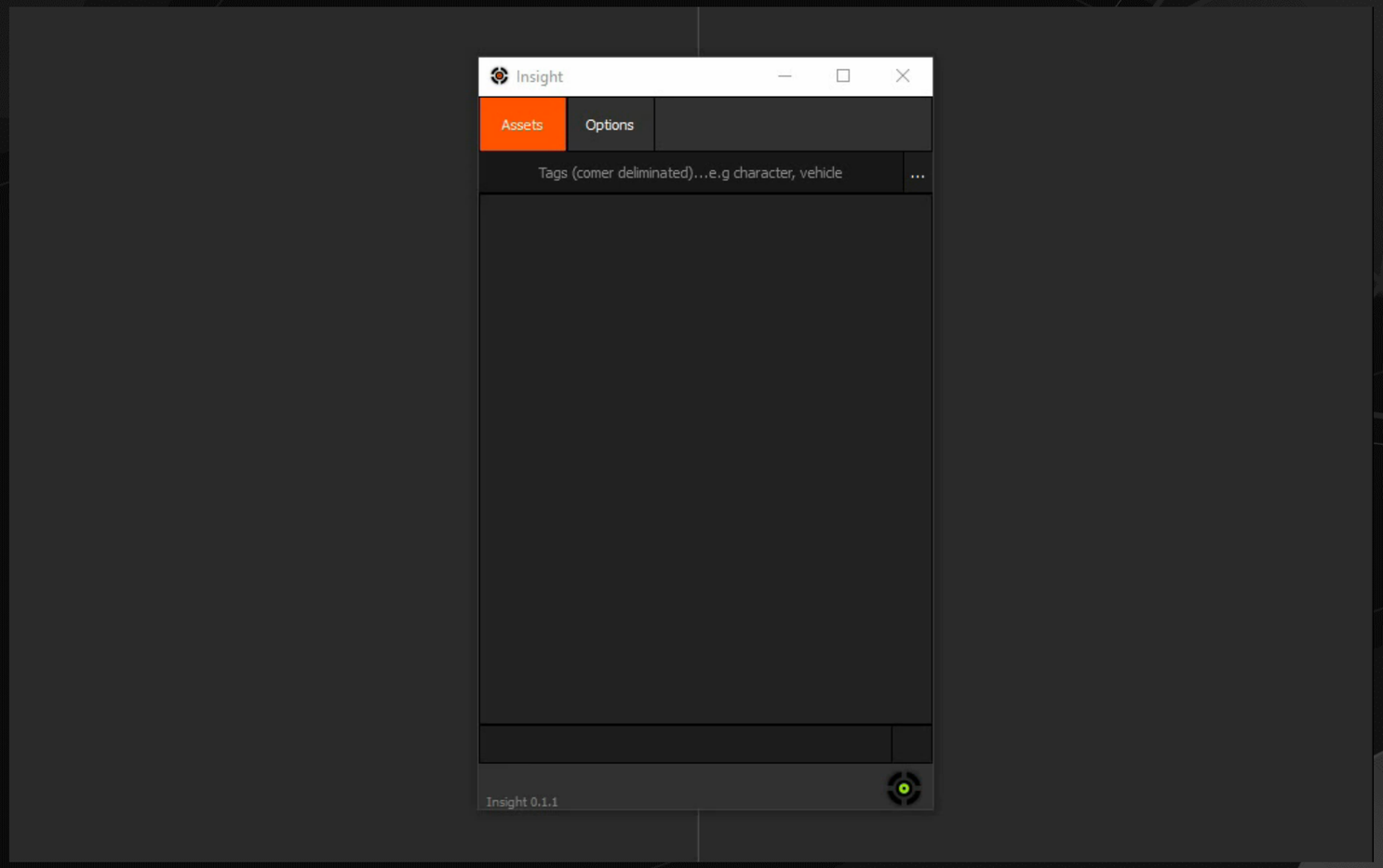
Detailed Blog: www.twisted.space

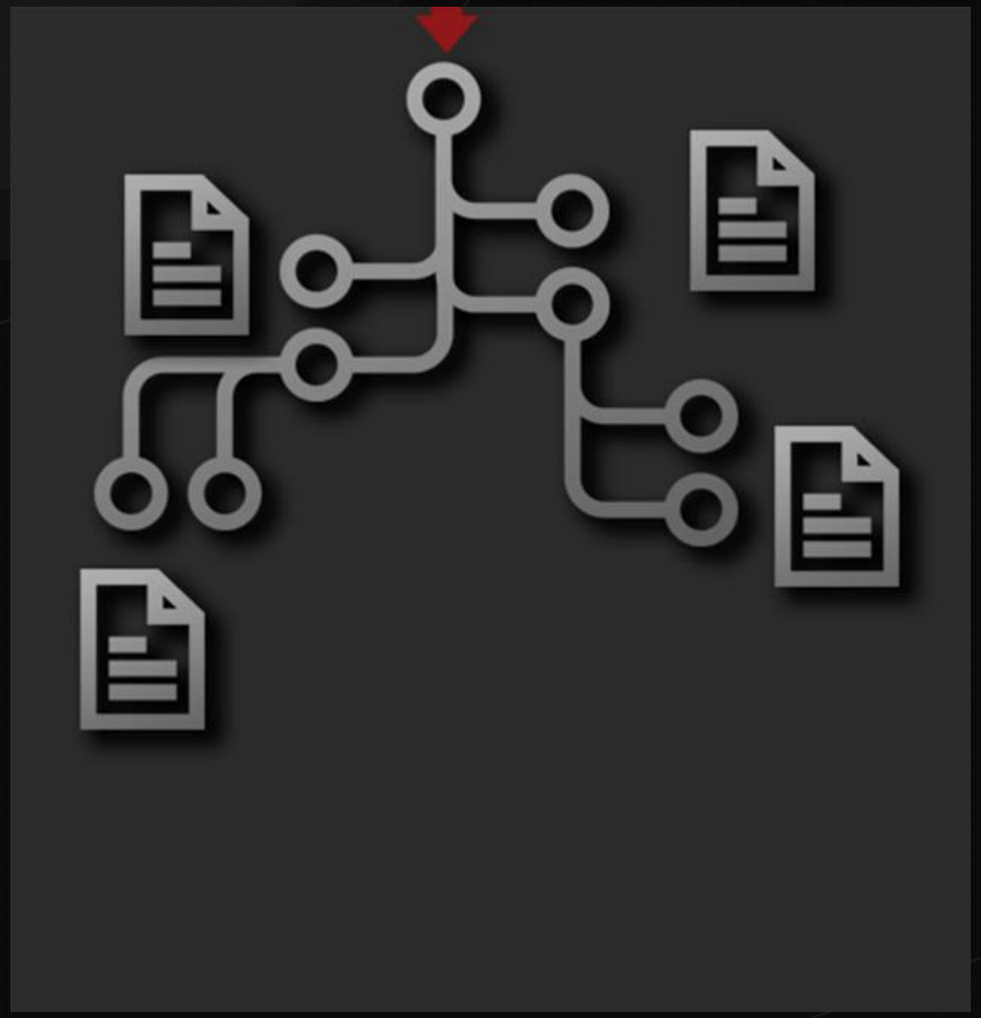

Assets
Options
rig
bird01b_rig.ma Have Latest
bird02_rig.ma
bird03_rig.ma Have Latest
blackark01_rig.ma
bloodcauldron01_rig.ma Have Latest
boar01_rig.ma Have Latest
boltthrower01_rig.ma Have Latest
boltthrower01b_rig.ma Have Latest
boltthrower02_rig.ma Have Latest
Insight 0.1.1

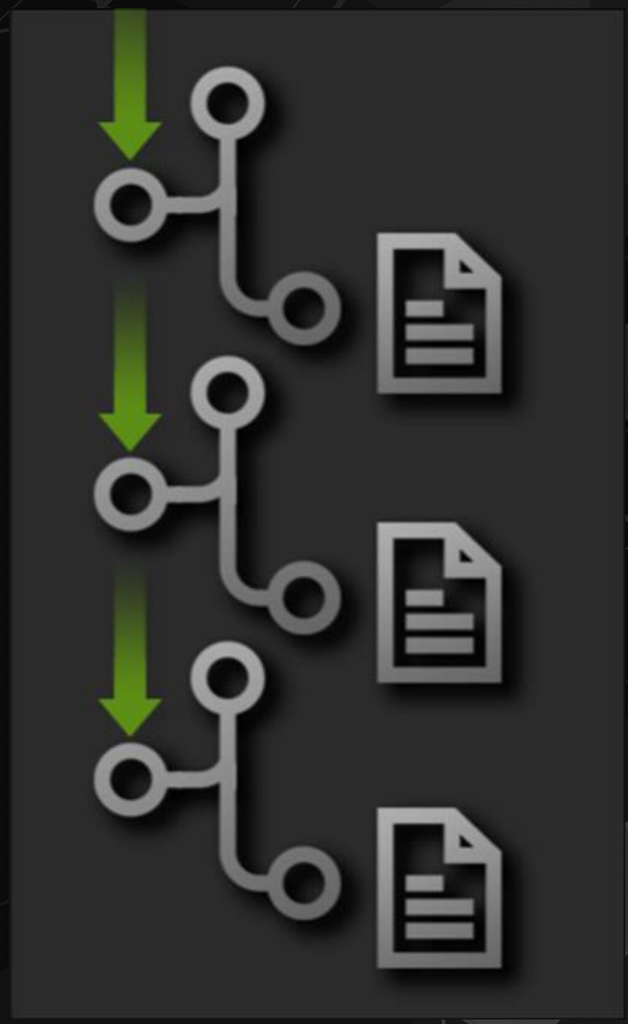

JSON
JSON


Run:
scratch_181
scratch_179

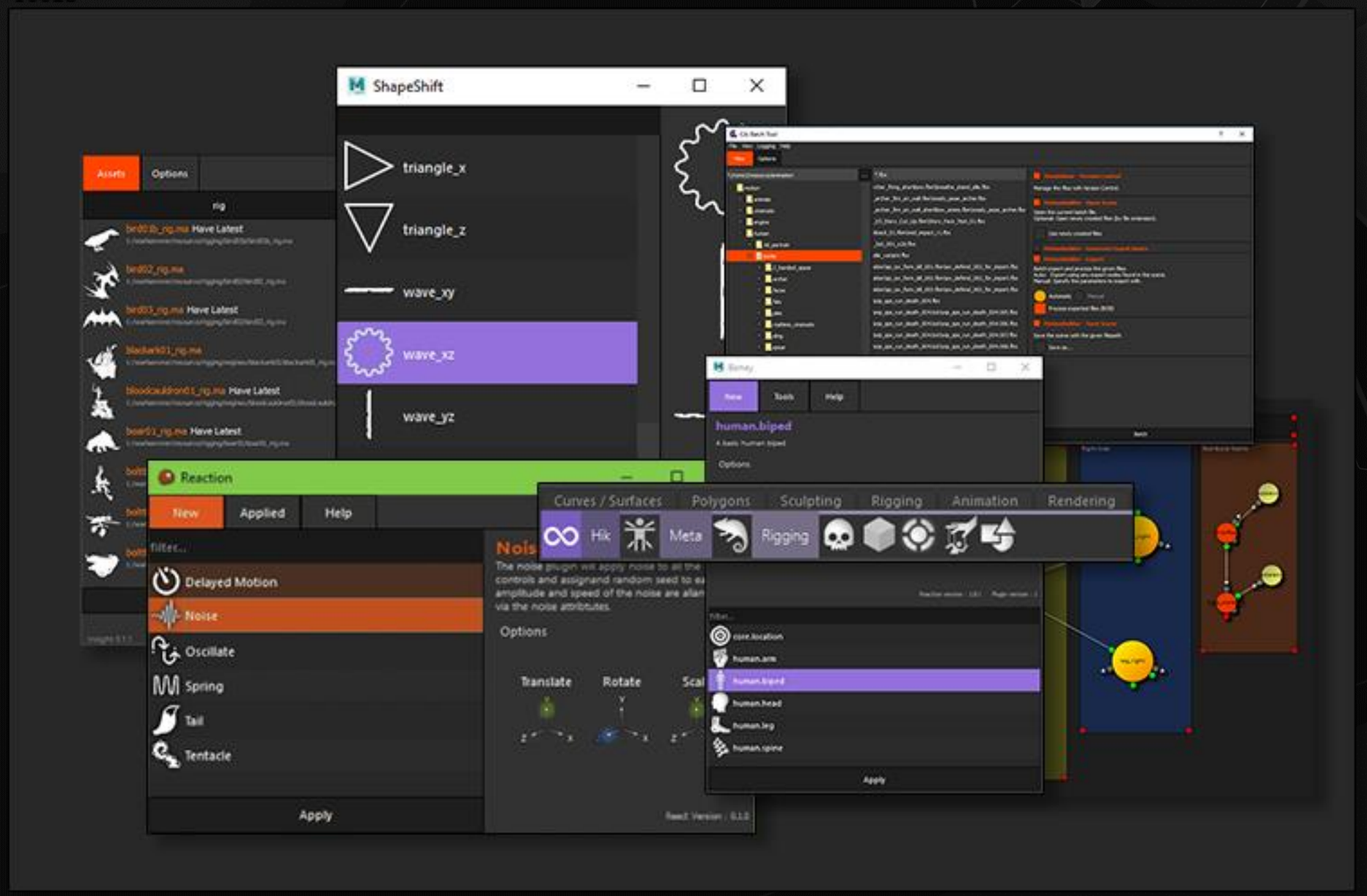

Insight
Assets
Options
saur
Yuanmousaurus jiangyiensis
Yunnanosauridae
Yunnanosaurus
Yunnanosaurus huangi
Yunnanosaurus robustus
Yunnanosaurus youngi
Zapalasaurus
Zapalasaurus bonapartei
Zigongosaurus
Zizhonosaurus
Insight 0.1.1


Insight
Assets
Options
Tags (comer deliminated)...e.g character, vehicle
...
Insight 0.1.1





Tools



Libraries

FBX PROCESSOR

VCS MANAGER

TW EXPORT

RIG CONFIG

Add functionality without adding complexity

Applicable to many frameworks and toolsets

Promotes code consistency

Quicker implementation times for functionality

Encourage creativity and experimentation


GDC
Mike Malinowski
Lee Dunham
Toby Harrison-Banfield
Joe Hornsby
Jodie Azhar
Mark Perry
Nathan Lawrence
CREATIVE ASSEMBLY™
GAME DEVELOPERS CONFERENCE® | MARCH 19-23, 2018 | EXPO: MARCH 21-23, 2018 #GDC18
UBM