MaterialX Nested Nodegraphs

This document outlines the proposal to support a `<nodegraph>` which a child of another `<nodegraph>`. As there are no syntax changes, the proposal is for a **1.38.x** incremental release.

Authors

Niklas Harrysson, Bernard Kwok, Jonathan Stone

Motivation

Currently compound `nodegraph`s can be specified at the document level. The extension to this is to allow nesting at **any** level in a graph hierarchy.

This allows for packaging of sub-parts of a graph into logical units for better organization and readability. When a graph becomes large it becomes convenient to have the ability to wrap parts of it into compounds where it make sense.

For example, in the nodegraph for `standard_surface` there are several parts of the graph that can be separated into child compounds that calculate a particular piece of logic such as:

- Tangent rotations
- How coat affect roughness
- How coat effect diffuse and sss colors
- How fresnel effects the EDF

Even the logical layering of BSDFs can probably have been separated out into compounds for better readability in a graph editor.

As an artist this becomes a useful tool in the toolbox for organizing graphs.

In addition, when such a compound is found to be reusable it can be published as a new definition (`nodedef`) with a functional nodegraph allowing it to be instantiated and reused. Without the ability to create child compounds an artist currently has to first move the graph up to the document level, then recreate the logic there, and then perform the "publish" to a nodedef.

Another motivation is that a user is presented with the ability to
construct nested nodegraphs in the Graph Editor, but cannot save the result to a document.

Syntax Changes

None

Connection Logic Changes

Connection handling becomes more consistent between nodes and nodegraphs by
generalizing connections. This is achieved by removing the restriction that some things you can do on nodes are not allowed on nodegraphs. Namely:

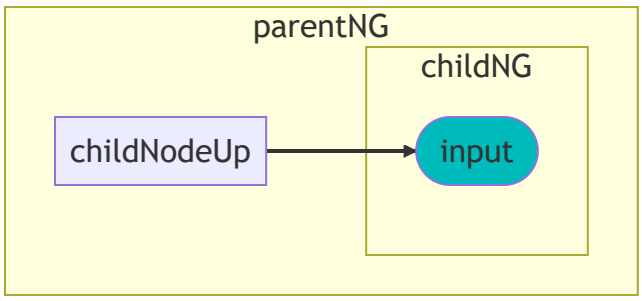
- `interfacename` can be specified on nodegraph `<input>`s to allow connections between inputs on a child nodegraph and a parent nodegraph.

Possible Configurations

Given a parent `<nodegraph>` called `parentNG` and a child nodegraph `childNG`, and child nodes `childNodeUp` and `childNodeDown`:

1. The `<input>` on `childNG` is connected to the output of an upstream node `childNodeUp`

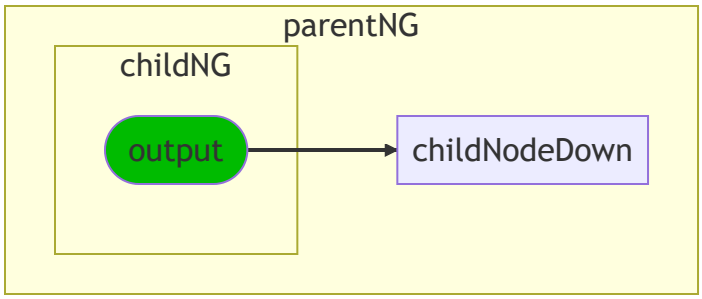
```
<input ... node="childNodeUp"> [output="output on childNodeUp"]>
```



The `output` is specified if `childNodeUp` has multiple outputs.
If the upstream node was a nodegraph then syntax would be `nodegraph=`
instead of `node=`. (See *)

2. The `<output>` on `childNG` is connected to the input of a downstream node `childNodeDown`

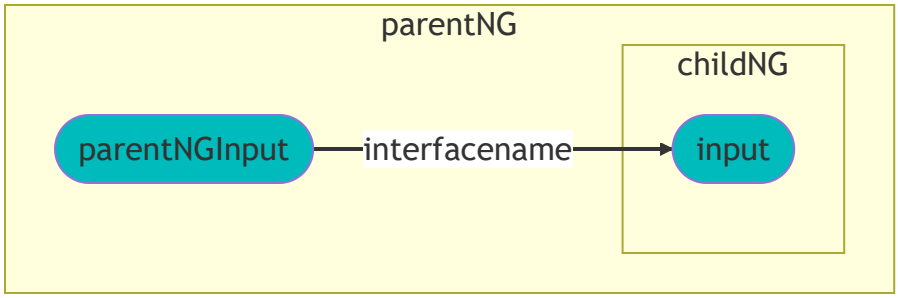
```
<input ... nodegraph="childNG"> [output="output on childNG"]>
```



The `output` is specified if `childNG` has multiple outputs.

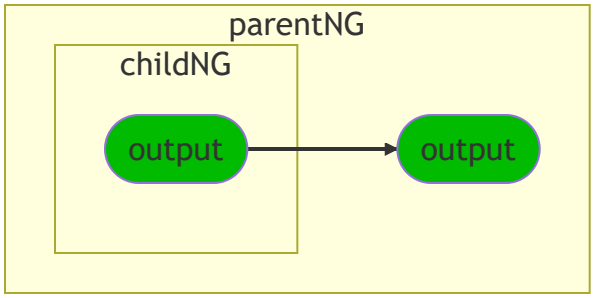
3. The `<input>` on `childNG` is connected to the interface `<input>` called `parentNGInput` of the parent nodegraph `parentNG`

```
<input ... interfacename="parentNGInput"`>
```



4. The `<output>` on `childNG` is connected to the `<output>` of the parent nodegraph `parentNG`

```
<output ... nodegraph="childNG" [output="output on childNG"]>
```



with the `output` being specified if `childNG` has multiple outputs.

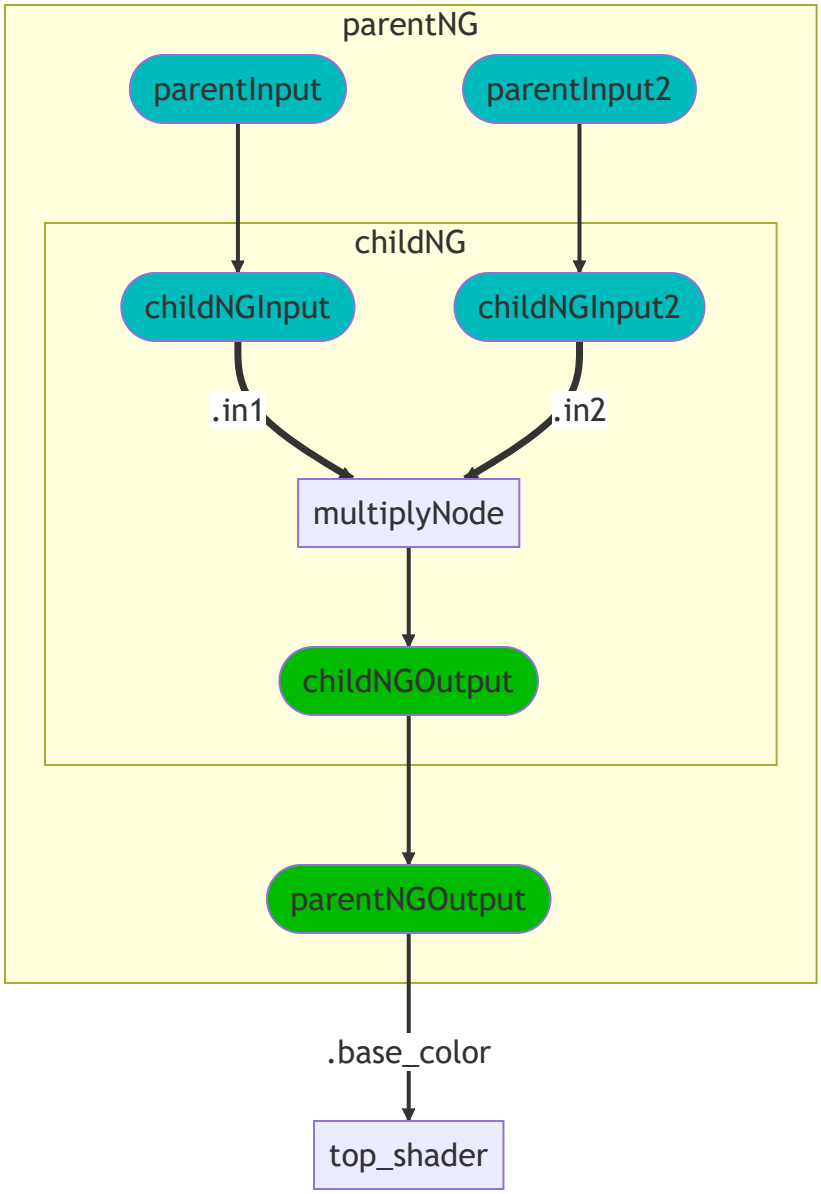
(*) Note that `<nodegraph>` to `<nodegraph>` connections are already supported so is similar to the node to nodegraph connection scenario.

Interface Example

Example shows interfaces being connected between parent and child `nodegraph`s

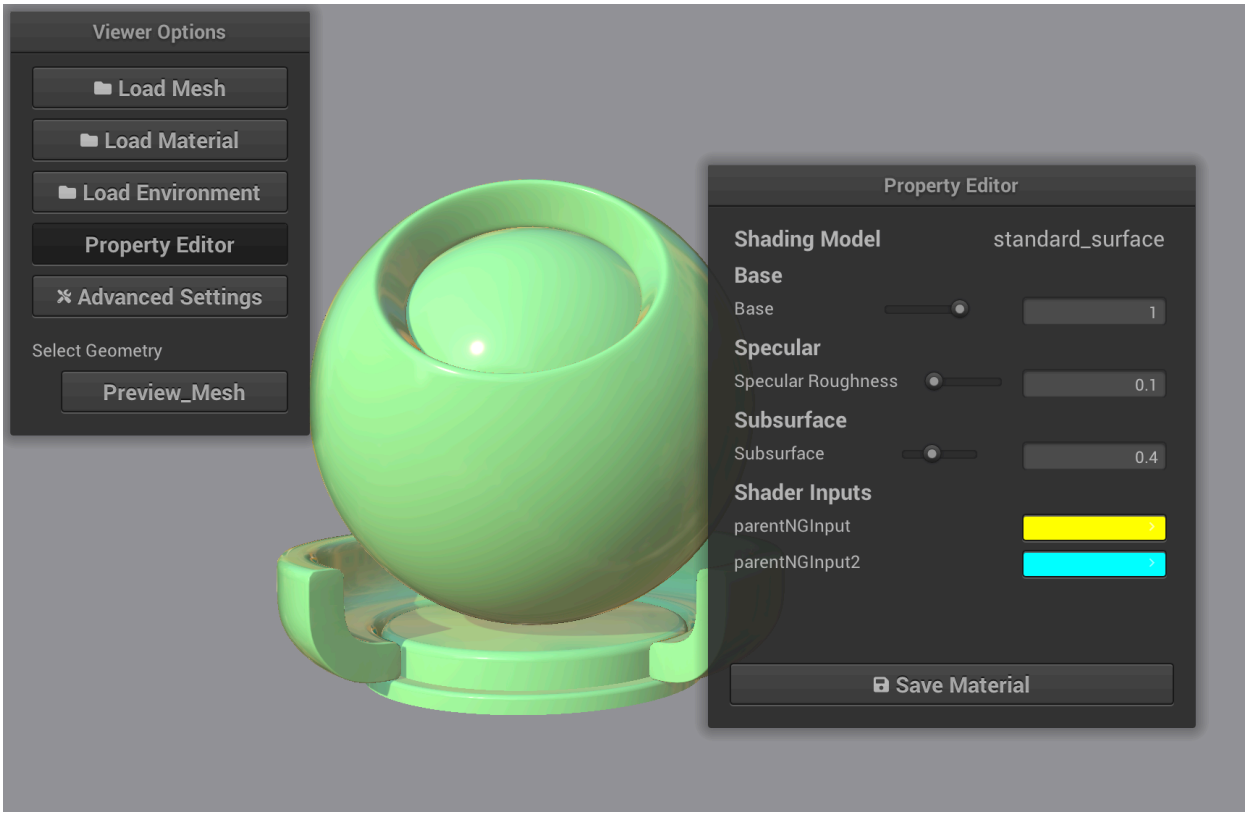
```
<?xml version="1.0"?>
<materialx version="1.38" colorspace="lin_rec709">
  <nodegraph name="parentNG">
    <input name="parentNGInput" type="color3" value="1, 1, 0.2"  />
    <input name="parentNGInput2" type="color3" value="0.2, 1, 1" />
    <nodegraph name="childNG">
      <input name="childNGInput" type="color3" interfacename="parentNGInput" />
      <input name="childNGInput2" type="color3" interfacename="parentNGInput2" />
      <multiply name="multiplyNode" type="color3">
        <input name="in1" type="color3" interfacename="childNGInput" />
        <input name="in2" type="color3" interfacename="childNGInput2" />
      </multiply>
      <output name="childNGOutput" type="color3" nodename="multiplyNode" />
    </nodegraph>
    <output name="parentNGOutput" type="color3" nodegraph="childNG" />
```

```
  </nodegraph>
</materialx>
```



The resulting rendering in MaterialXView would look something like this,
where the parent nodegraph inputs (`parentNGInput`, `parentNGInput2`) are exposed as input uniforms via shader code

generation.



Implementation Requirements

- Graph traversal logic, shader generation, and value evaluation handles `interfacename` on nodegraph inputs, and traversal from a parent nodegraph `output` to a child nodegraph `output`
- "Inherited" properties (such as `fileprefix`, `colorspace`) evaluate properly through parent/child `nodegraph`s. This should already be the case.
- No "upgrade" path is required.