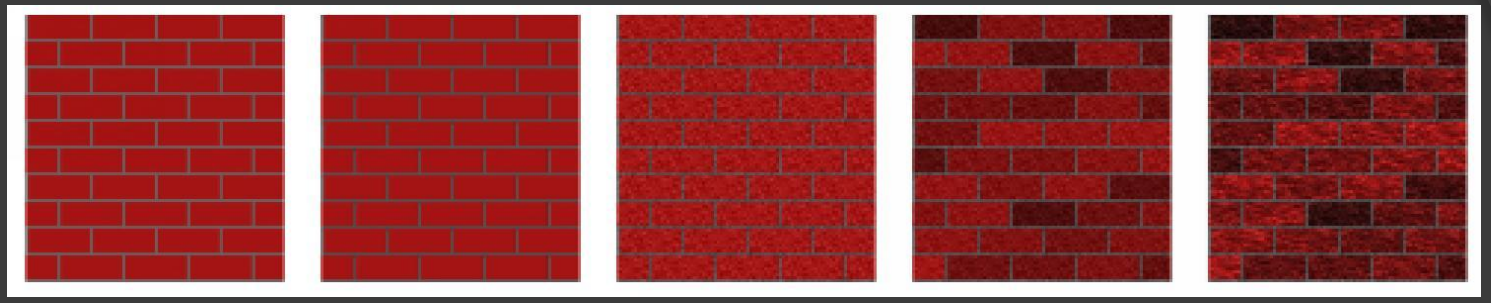
BRICK SHADER IN MAYA

Goal

Brick Criteria



- Simple brick pattern defined by solid colors for the mortar and bricks
- Indented mortar
- Graininess throughout the whole image
- Variations in color from brick to brick
- Color variations within each brick

Paint Criteria

- Painted areas defined by a solid color
- Variation in color within the painted areas
- Bumpiness within painted areas to simulate old, peeling paint

Challenges

- Maya C++ API
- Variation in Brick and Paint Color
- Simulation of Mortar Indentation and Paint Bumpiness

Maya API

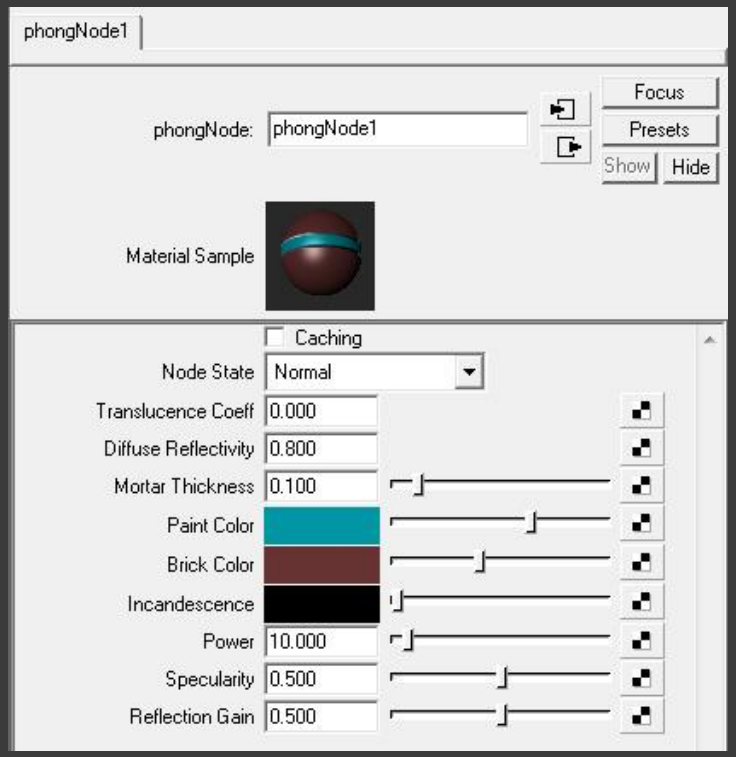
Maya, at its core, is a dependency graph. It allows you to feed arbitrary data into a series of operations to produce an output.

The data and their operations are called nodes within the dependency graph.

Using the Maya API

- Create a node that takes a number of input attributes and then outputs a color attribute.
  - The input attributes can be user-defined - such as brickColor, or pre-defined - such as normalCamera. Maya knows how to provide your plug-in with information from the scene that corresponds to the pre-defined attributes.
  - The output attributes can also be user-defined, or pre-defined. However, pre-defined (color, glow color, displacement, and more) are usually used because Maya knows how to connect these outputs to shading groups within your scene.

- Create a template so that the user-defined input attributes show up in the UI.
  - The template is simply a .mel file with an ‘AE’ prefix.




```
global proc AEphongNodeTemplate( string $nodeName )
{
    AEswatchDisplay $nodeName;
    editorTemplate -beginScrollLayout;

    editorTemplate -beginLayout "Common Material Attributes" -collapse 0;
        editorTemplate -addControl "paintColor";
        editorTemplate -addControl "brickColor";
        editorTemplate -addControl "mortarThickness";
        editorTemplate -addControl "incandescence";
        editorTemplate -addControl "diffuseReflectivity";
        editorTemplate -addControl "translucenceCoeff";
    editorTemplate -endLayout;
```

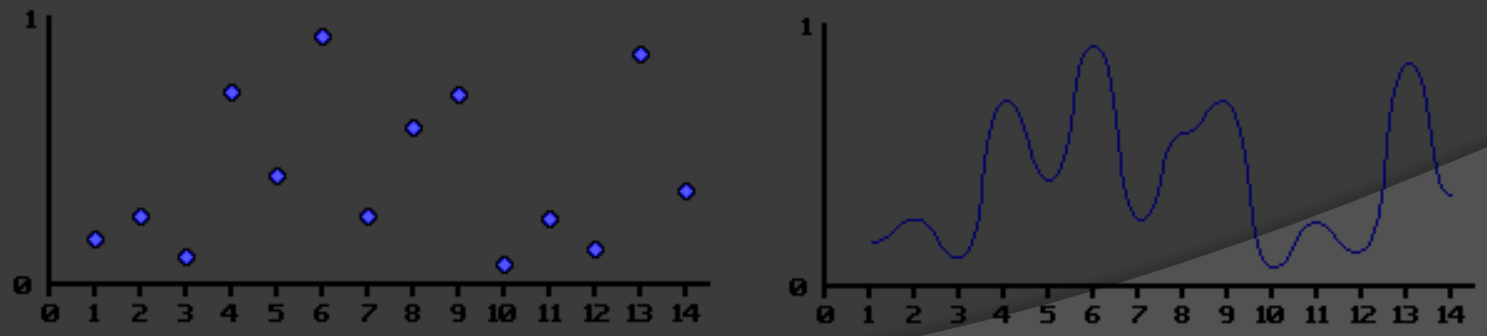
- Initialize user-defined and pre-defined attributes for use in later computation

```
MStatus PhongNode::initialize()
{
    MFnNumericAttribute nAttr;
    MFnLightDataAttribute lAttr;

    aTranslucenceCoeff = nAttr.create("translucenceCoeff", "tc",
                                      MFnNumericData::kFloat);
    MAKE_INPUT(nAttr);

    aDiffuseReflectivity = nAttr.create("diffuseReflectivity", "drfl",
                                        MFnNumericData::kFloat);
    MAKE_INPUT(nAttr);
    CHECK_MSTATUS ( nAttr.setDefault(0.8f) );

    aBias = nAttr.create( "mortarThickness", "b", MFnNumericData::kFloat);
    MAKE_INPUT(nAttr);
    CHECK_MSTATUS(nAttr.setMin(0.0f));
    CHECK_MSTATUS(nAttr.setMax(1.0f));
    CHECK_MSTATUS(nAttr.setDefault(0.1f));
```

- Most of our implementation will be done in the compute function, which is considered the “brain” of all nodes.

Perlin Noise

- Can implement many textures: smoke, wood, marble, etc. or in our case, the variation in bricks and paint
- The improved algorithm, 6x^5 – 15x^4 + 10x^3 is used to smoothly interpolate between points in 1-, 2- or 3-D space

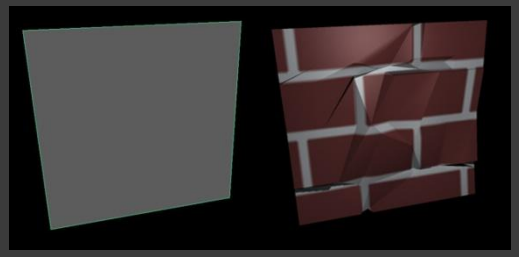


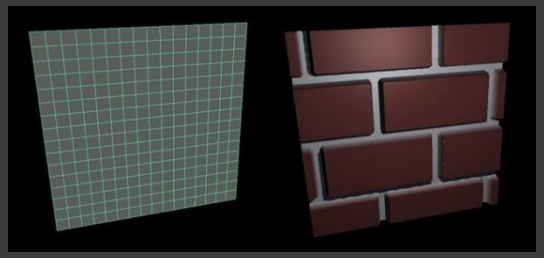
Perlin Noise cont.

- Uses a pseudo-random number generator
  - Takes an integer and returns a random number based on that parameter
  - The same parameter returns the same random number

Simulation of Brick Depth

- Bump map and displacement are both options.
  - Displacement is one of the pre-defined outputs so calculating it is very easy with the Maya API.
  - However, displacement requires a lot of extra vertices.

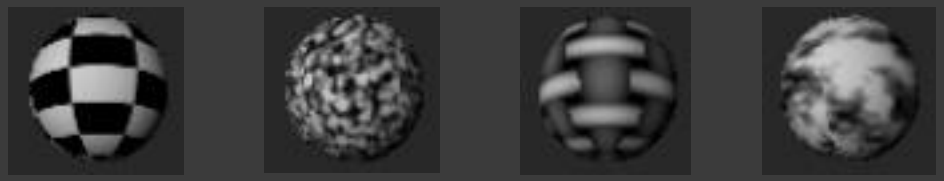




  - Bump mapping can be calculated by editing the normal and thus the pre-defined output color. The normal in camera space, and the tangents in both the U and V directions are pre-defined input attributes, so Maya automatically provides their values per pixel.

Existing Solutions

- We've been unable to find a completely procedural brick shader for Maya. However, there are some examples of procedural shaders in /Program Files/Autodesk/devkit/plug-ins.



Risks

- Achieving realistic-looking bump mapping for mortar indentation; especially since we probably won't have cast shadows between the bricks.
- Mimicking the large amount of variation in color and texture found in a real brick wall may require a lot of complex code.