glTF MaterialX USD Interop

The following are notes for discussion which involves glTF, MaterialX and USD

Background

1. USD / glTF / MaterialX interactions require a clear understanding of workflows and the roles each component plays within those workflows.
2. glTF is a "last-mile" format so what a *glTF -> MaterialX*, or *MaterialX -> glTF* workflow while compelling to have, we should carefully consider what this entails in terms of *data representation* and *data "loss"* (for lack of a better word).
3. The glTF -> MaterialX (and USD) workflow can be mostly non-lossy with the current MaterialX *glTF pbr node* definition (which when imported can be a USDShade node).
    a. This allows representation of glTF pbr plus directly connected upstream images+placement to be accessible in various authoring / lookdev / rendering worklows.
    b. Also *code generation* for consistent conversion to target shading languages now possible. (glTF direct implementations does not inherently guarantee any kind of render implementation consistency.)
4. The MaterialX -> glTF workflow requires "distillation" and so far current prototype implies that this can always be can be lossy (Autodesk).
    a. One vector for loss is that different shading models must be translated to glTF PBR.
    b. A second vector is arbitrary shading graphs can be created in MaterialX which must be "baked" somehow.
    c. All meta-data is lost.
5. You can run into this with USD where it can be "worse" via custom schemas.

Points of Discussion

1. **MaterialX as an extension**.
    a. Similar to the MDL proposal
    b. The key difference is the proposal would *not* to keep 2 representations of the same material but to directly support a MaterialX atomic "node".
    c. The MaterialX schema defines the inputs and outputs in the data model versus this being part of code logic.
    d. Implementation toggles are not part of the definition.
2. There are concerns that this "breaks" the mandate of the glTF format as MaterialX becomes an external dependency which requires more "complex" handling.
    a. This is already required but is provided by multiple different libraries (e.g. there is no "one" parser / evaluator for the glTF format).
    b. There is the idea to just have **MaterialX in JSON** format but this misses the requirement for graph *value resolution* (which is similar but less extensive in MaterialX than what is required for USD resolution).
    Usage of many "hand-made" JSON parsers can be appealing but can lead to inconsistency quickly.
    c. Pre-evaluation is possible here. (*)
3. Code generation available but there are concerns about ***shader performance*** and *duplication of resources*. These are generally valid concerns.
    a. There is inherent complexity in the shaders as fidelity is the initial goal. Refinements can made over time.
        i. Fidelity *options* are being discussed on the MaterialX side with optimizations for IBL sampling code already proposed.
    b. *Performance metrics* would be worthwhile to obtain in general (not just for Web deployment).
    c. Again pre-evaluation is possible here. (*)
    d. For resources, MaterialX does no manage resources. However If multiple formats or non-performant resource formats are explicitly being used then this should be looked for alternatives. This can be part of the shader translation, utility or core definition libraries. e.g Support for cubemap lookups for IBLs might be worthwhile as a new core definition.
4. (*) **Pre-evaluation**:
    a. A small subset of "essential" MaterialX shader nodes / graphs can be considered.
        i. This is interest to IKEA and Autodesk, but could also be to others.
        ii. The better the match between runtime and authoring definitions – the better the interop.
    b. *It seems worthwhile to discuss what this set is*.
    c. The converse of adding encapsulations of glTF upstream mappings as new definitions is already being considered. e.g. for Occlusion /Roughnes/Metal channel extraction / combining which helps the distillation process but could be a generally useful utility for image data packing/unpacking – perhaps even for output packing for things like AOVs.
    d. There is the "node system" in Three.js as well which can be looked at.
        i. Work on this can be found here: https://github.com/mrdoob/three.js/issues/20541
        ii. The idea of JSON bindings, and approach to pull pull out code for MaterialX "equivalent" nodes can be found in this discussion.
    e. USD definitions consume MaterialX definitions so definition / interface synchronization is already considered. This is not currently the case even with the single glTF pbr node.
    f. From a shader point of view there has been discussion about pre-generating code blocks which can be glued together for reuse. This may or may not be something which requires formal definitions but could be done on the fly.