OpenMfx: An API for cross-software non-destructible mesh effects

Élie Michel
LTCI, Télécom Paris, Institut Polytechnique de Paris
Palaiseau, France
elie.michel@telecom-paris.fr

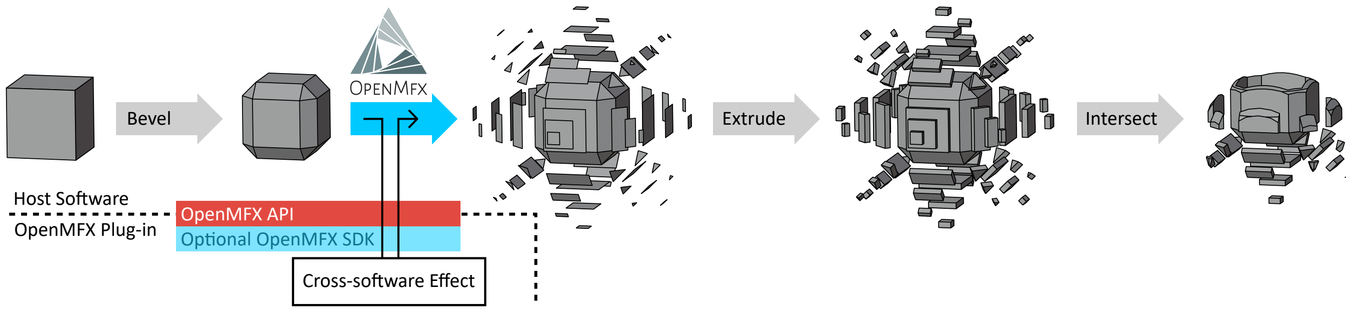

Figure 1: An example of sequence of non-destructive effects transforming a simple editable base mesh (e.g. a cube) into a more complex asset. Operations like bevel, extrusion or boolean intersection are available in most of common mesh-based 3D modeling suites, but the effect applied at the second step is less common. By implementing it as an OpenMfx plug-in, it is available in any supporting host software and thus the asset is interexchangable in its non-destructive form.

CCS CONCEPTS

• **Computing methodologies → Graphics file formats**; **Mesh geometry models**; • **Software and its engineering → API languages**.

KEYWORDS

Interoperability, Mesh Modeling, Open Source, Plug-in API

ACM Reference Format:
Élie Michel. 2021. OpenMfx: An API for cross-software non-destructible mesh effects. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Posters (SIGGRAPH '21 Posters), August 09-13, 2021.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3450618.3469168

1 INTRODUCTION

Non-destructive operations are widely used in mesh-based 3D modeling to add procedural effects on top a coarse geometry while allowing it to remain editable. Common such operations include surface subdivision, beveling, repetition, boolean operations. Combined together as a stack or even as a direct acyclic graph, they are a very powerful tool to build parametric assets. Although this mechanism is present in many different 3D modeling suites (Maya, Houdini, Blender, Cinema4D, 3ds Max, etc.), it is not easy to share a parametric asset across them. Indeed, they do not all implement the exact same set of operations, and even when they do, there might be slight discrepancies in their behavior. We designed a plug-in API that enables one to have all supporting 3D modeling suites share the same implementation of a given non-destructive effect. It thus becomes possible to share scenes featuring non-destructive effect without having to destructively bake them.

2 TECHNICAL APPROACH

We build on top of OpenFX [Association 2006], an industry standard plug-in API that has been developed by Foundry while facing a very similar problem in the field of compositing, which is nothing else than non-destructive image editing. OpenFX has been designed from the ground up with modularity in mind so we were able to fully reuse its core, including the base plug-in mechanism and the API for setting effects' parameters. Our mesh effect API is introduced as an extension next to its image effect API. In the end, most of the decisions we have made were related to the representation of meshes that transits through the API.

This representation aims at supporting a wide variety of mesh topology, including n-gons of arbitrary point count, unconnected points (for point clouds), loose edges (for wireframe meshes) and mixes of all of these. It aims at a minimal enough memory footprint, meaning that the representation should be non-redundant and also that it should be able to point to wherever the data already is in the host's memory rather than copying it, if possible. And it aims at a simple design, to avoid dealing with multiple particular cases.

Attributes. An OpenMfx mesh is then simply a list of attribute buffers. A given attribute is relative to either points, faces or face corners and contains between 1 and 4 values of a given type (short, int or float) for each point/face/corner. For instance the vertex positions are given by a 3-component float point attribute. The connectivity information makes no exception, it is given by attributes, namely an integer face attribute telling for each face its number of corners and an integer corner attribute telling for each corner the index of the point it refers to. These are equivalent to what USD's `UsdGeomMesh` class calls resp. `faceVertexIndices` and `faceVertexCounts`, and is also close to Blender's and Houdini's representations[1].

Note that we decided not provide edge attributes. Edges that belong to no face, called "lose edges" are listed as two-point faces, other edges are omitted because implicitly defined by faces. Since edges are shared across faces, the price in clarity to include edge attributes was too important compared to the fact that many existing API don't support them anyways.

Attribute buffers are of two kinds. Some own their data, which is freed when the mesh is released. Others are non-owner attributes, meaning that they point to an existing memory location, that is assumed to remain valid throughout the execution of the effect. The memory layout is described by a flexible *buffer protocol*, loosely inspired by Python's [Python [n.d.]], such that the host can use non-owner attributes as much as possible to feed mesh attributes to the effect. Only attributes whose layout on host side cannot fit the buffer protocol need to be copied. Non-owner attributes can also be used in outputs, for instance to forward unchanged input attributes without copying them at all.

On-demand data. To alleviate further the need for memory transfers, an effect must explicitly request the attributes it needs. A requested attribute is assigned a *semantic* flag hinting about the meaning of the attribute (color, normal, texture coordinate, weight) and that the host may use to suggest the user which host-side attribute to feed as the requested one.

Another information that is provided only on demand is the world to local transform matrix associated to the mesh. It is not available unless requested, such that the host's dependency graph can more finely avoid useless executions or dependency loops.

Specific Scenarios. While aiming at flexibility, it was also important to ensure that some more specific yet very common cases are efficiently handled. Often the number of points per face is fixed. It might be to 3 (triangle only meshes), 4 (quad only), but also 2 (wireframe only). In such a case, a flag is used to avoid allocating a face point count buffer that would be uniform.

Attribute forwarding is already a good way to avoid unneeded copies of memory, but for the host's internal it might be useful to know even before running the effect that it will not change the connectivity of the mesh (e.g. a displacement effect). So called "deform only" effects can advertise this fact ahead of execution (at describe time) so that the host may handle them differently.

Limitations. Besides the aforementioned absence of explicit edge attributes, two features have been set aside compared to Maya's API. First OpenMfx does not allow faces with holes, i.e. faces that would be made of more than one loop of edges. This would abusively complicate the task of plugin writers while use cases are very uncommon. Secondly, there is no indirection between face corners and texture coordinate, so no proper definition of "UV island".

[1]Houdini calls "vertex" what we call "corner", but "vertex" used by Blender to mean what we and Houdini call "point", we settled on the less ambiguous term "corner".

3 FUTURE PROSPECTS

Currently, the OpenMfx API is getting stable and documented, one host is supported (a branch of Blender [Michel 2019b]) and another one (in Unity) is at the stage of proof of concept. Several effect packages are available, like *MfxVCG* [Michel 2019a] providing effects from *MeshLab*'s *VCGlib* [Cignoni 2008] or *MfxVTK* [Karabela 2020] providing effects from the *Visualization Toolkit*[Schroeder et al. 2006]. The ecosystem may still get improvements in order to ease its adoption. The current major limitation is lack of supporting hosts besides Blender and Unity, so we are working on a SDK similar to the one we have built to ease the creation of plug-ins but oriented towards the integration into new hosts. The Unity host is still limited at this stage as we have not defined a proper dependency graph yet.

In order to ensure harmonization among multiple implementations – which is sometimes a problem of OpenFX – we plan on providing an optional validation layer, reporting any inconsistencies in the use of the API. This will come with an overhead but would be used only at development time.

To meet our original goal of bringing the possibility to share scenes that have non-destructive effects and parametric shapes across software suites, we will integrate an USD schema [Studios 2016] telling which plug-in to instantiate where and with which parameters when exchanging a scene. Effect parameters would be driven by `UsdAttribute` and input mesh would be provided as `UsdRelationship`. The effect could request from their input the extra attributes provided through the `UsdGeomPrimvarsAPI`.

So far we have focused on time independent effects, but the original OpenFX API also supports time-dependent effects such as simulations, so by reusing their industry tested approach we could also support it for 3D meshes even though it may raise questions that are not our priority at the moment.

There is room in the design for augmenting the effects with new actions without breaking compatibility. A promising example would be to add the possibility to query the effect for differential information, e.g. to measure jacobian of the operation, to make it usable in contexts such as machine learning, differentiable rendering or inverse control [Michel and Boubekeur 2021].

ACKNOWLEDGMENTS

We would like to thank Tomáš Karabela for the insightful discussions we had to shape this API and arbitrate between design choices.

REFERENCES

The OpenFX Association. 2006. *OpenFX.* https://openfx.readthedocs.io

Paolo et al. Cignoni. 2008. MeshLab: an Open-Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference*, Vittorio Scarano, Rosario De Chiara, and Ugo Erra (Eds.). The Eurographics Association.

Tomas Karabela. 2020. *MfxVTK: An OpenMfx plug-in based on the Visualization Toolkit (VTK).* https://github.com/tkarabela/MfxVTK

Élie Michel. 2019a. *MfxVCG: An OpenMfx plug-in based on VCGlib.* https://github.com/eliemichel/MfxVCG

Élie Michel. 2019b. *OpenMfx for Blender.* https://github.com/eliemichel/OpenMeshEffectForBlender

Elie Michel and Tamy Boubekeur. 2021. DAG Amendment for Inverse Control of Parametric Shapes. *ACM Transactions on Graphics* 40, 4 (2021), 173:1–173:14.

Python. [n.d.]. *Python's Buffer Protocole.* https://docs.python.org/3/c-api/buffer.html

Will J Schroeder, Ken Martin, and Bill Lorensen. 2006. *The Visualization Toolkit (4th ed.).* Kitware.

Pixar Animation Studios. 2016. *Universal Scene Description.* https://graphics.pixar.com/usd