MEPP2: a generic platform for processing 3D meshes and point clouds

Vincent Vidal, Eric Lombardi , Martial Tola , Florent Dupont and Guillaume Lavoué

Université de Lyon, CNRS, LIRIS, Lyon, France

Abstract

In this paper, we present MEPP2, an open-source C++ software development kit (SDK) for processing and visualizing 3D surface meshes and point clouds. It provides both an application programming interface (API) for creating new processing filters and a graphical user interface (GUI) that facilitates the integration of new filters as plugins. Static and dynamic 3D meshes and point clouds with appearance-related attributes (color, texture information, normal) are supported.

The strength of the platform is to be generic programming oriented. It offers an abstraction layer, based on C++ Concepts, that provides interoperability over several third party mesh and point cloud data structures, such as OpenMesh, CGAL, and PCL. Generic code can be run on all data structures implementing the required concepts, which allows for performance and memory footprint comparison. Our platform also permits to create complex processing pipelines gathering idiosyncratic functionalities of the different libraries. We provide examples of such applications.

MEPP2 runs on Windows, Linux & Mac OS X and is intended for engineers, researchers, but also students thanks to simple use, facilitated by the proposed architecture and extensive documentation.

CCS Concepts

• ***Computing methodologies*** → *Mesh models; Point-based models;* • ***Software and its engineering*** → *Software libraries and repositories;*

1. Introduction

With the increasing capability of 3D data acquisition devices, modeling software and graphics processing units, three-dimensional (3D) graphics are now commonplace in many applications from simulation and gaming to upcoming tele-immersive communication. 3D assets commonly consist of surface meshes on which texture images are mapped (e.g. affecting surface geometry or normal, or reflectance terms such as diffuse, gloss, specular). Another popular 3D representation is the 3D point cloud that has the benefit to be the direct output of acquisition devices.

To ease the development of new algorithms and stimulate research around the processing of 3D mesh and point cloud, it is crucial to have efficient open source tools and libraries available for the scientific community. This need for dedicated libraries is emphasized by the fact that manipulating 3D data is far more complex than manipulating 2D images. As an example, obtaining the direct neighborhood of a pixel in an image is a trivial operation; however, computing neighbors of a 3D vertex in a 3D mesh (resp. point cloud), with reasonable time complexity, requires a dedicated data structure such as halfedge (resp. octree). In this context, we introduce the MEPP2 platform[†], a C++ software development kit and GUI for processing and visualizing 3D surface meshes and point clouds.

Several platforms exist for processing 3D meshes such as Meshlab [CCC08] based on VCGlib[‡], MEPP [LTD12], and libigl [JP*18]. These platforms either use their own data structures or are based on existing ones such as OpenMesh [BSBK02] or CGAL Polyhedron [The20]. For point clouds, a popular platform is CloudCompare [Gir19], and available libraries include PCL [RC11] and CGAL Point Set. The MEPP2 platform differs from these existing tools in several ways:

It supports a wide range of 3D data: static and dynamic meshes and point clouds, together with several types of attributes: vertex/point colors and normals, different kinds of texture maps (including physically-based maps). It is not limited to one single data structure but integrates a wide range of them (for both mesh and point cloud representation): OpenMesh, CGAL Surface Mesh, CGAL Polyhedral Surface, CGAL Linear Cell Complex, AIF (Adjacency and Incidence Framework), PCL (Point Cloud Library), and CGAL Point Set. It offers generic-programming abstraction layers, allowing to invoke any data structure with the same source code. It offers an application programming interface (API) for creating new process-

† https://github.com/MEPP-team/MEPP2

‡ https://github.com/cnr-isti-vclab/vcglib/

DOI: 10.2312/egs.20201010

ing filters and a graphical user interface (GUI) to integrate filters as plugins. It is developer-oriented and provides extensive documentation and tutorials for creating filters and plugins.

The next section introduces the architecture of MEPP2. Section 3 provides source code examples, while currently available filters are presented in Section 4. Section 5 details two applications.

2. Architecture

Figure 1 shows the architecture of the MEPP2 platform. The core of the platform is the central layer: the FEVV template library (FEVV holds for *Face Edge Vertex Volume*). It relies on a set of concepts [GJS*06], which provide an abstraction layer over several third party mesh and point cloud data structures. Generic filters can then be created based on this template library. Structure-specific filters can also be written. Filters can either be called using a command-line interface or integrated into the graphical user interface. All these elements and principles are detailed below.

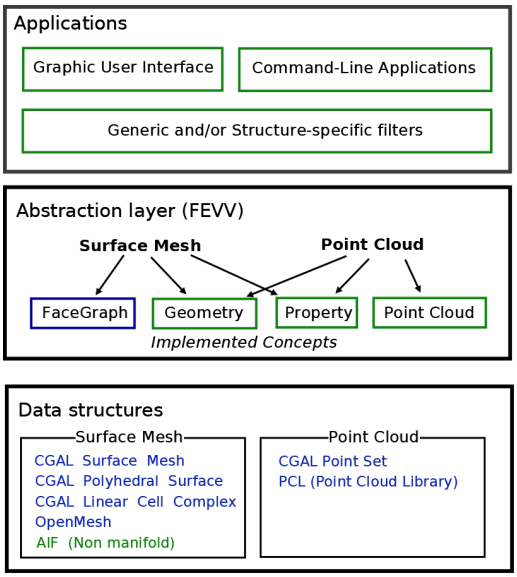

Figure 1: *Layers overview of the MEPP2 platform. Third-party concept and data structures are represented in blue, while MEPP2 contributions are depicted in green.*

2.1. Concepts

In C++, the development of a generic *-highly reusable-*, flexible and efficient software library is done through the use of constrained template parameters that are named concepts [GJS*06]. This way of programming is as powerful as inheritance with polymorphism programming but more efficient in running time. In the source code of a generic function, these constrained template parameters are named with a concept type that explicitly refers to a set of requirements (valid expressions, associated types, etc.). A data structure that implements all the concept's prerequisites can then be used as input of a generic function (for the corresponding parameter).
In the MEPP2 platform, Boost[§] and CGAL libraries provide the main concepts to process the topology of a surface mesh (*FaceGraph* in Fig. 1). Boost defines the concepts needed to process a graph (e.g. boost::Graph, boost::IncidenceGraph, etc.) and CGAL defines concepts that refine the boost::IncidenceGraph concept to process a graph that is also a polygonal 2-manifold surface. Beyond this use of the CGAL *FaceGraph* concept, MEPP2 supplies new concepts for geometry, point cloud, and property maps (this latter is extended from Boost). These concepts are detailed below.
Boost graph concept. It provides vertex and edge manipulation features, such as vertex/edge descriptor, vertex/edge iterator and some basic functions to iterate on vertices and edges of the graph.
CGAL HalfedgeGraph and FaceGraph concepts. They extend the Boost graph concept with halfedge and face related features. CGAL also offers circulators around a face or a vertex, which are implemented using HalfedgeGraph concept.
FEVV geometry concept. This concept mainly provides a generic way to create a point and to access its x, y, and z coordinates. It also gives access to Point, Vector, Scalar, and Kernel associated types.
FEVV property map concept. The original need for "generic" property maps arises from contexts where one needs to have the warranty that storing/accessing a property map associated with some mesh will function even if such a property map is not natively supported by the specific mesh data structure. For example, when using the native readers, the properties read from the mesh file (like vertex color, vertex normal, face color...) are stored inside the data structure itself. But when using a generic reader, we do not have anymore access to the data structure internal storage. "Generic" property maps, based on boost property maps, provide such a location for storing mesh properties independently from any specific data structure. The FEVV property map concept also provides a property map container that is used to gather all the property maps associated with the same mesh.
FEVV point cloud concept. The point cloud concept is intended to manipulate point cloud objects generically. It reuses the notations of the Boost graph concept related to the vertices (for access, addition, and removal) so that a simple algorithm iterating on vertices can be written in a generic way for meshes and point clouds. The point cloud concept also provides a k-nearest neighbor feature, which is common in point cloud processing.

2.2. Adjacency and Incidence Framework (AIF)

MEPP2 provides an implementation of the Adjacency and Incidence Framework (AIF) data structure [SG03] to represent and process non-manifold surface meshes. AIF captures the incidence relationships between faces and edges, and between edges and vertices. Our AIF implementation follows the boost::BidirectionalGraph and CGAL::FaceGraph/CGAL::MutableFaceGraph concepts, but does not fulfill the CGAL::HalfedgeListGraph since it does not store halfedges. Topology helpers are provided to ease access and modification of incidence relationships, to evaluate some topological predicates onto vertices, edges, faces, and meshes such as manifoldness, and to implement an HalfedgeGraph wrapper.

2.3. Graphical user interface (GUI), rendering and plugins

The platform provides a modular architecture through the use of processing filters available as dynamic plugins (.dll for Windows/.so for Linux/.dylib for Mac OS X) with plugins selection at

§ https://www.boost.org/

compilation time followed by their detection and automatic loading at runtime. A plugin easily wraps a filter (a skeleton plugin with a simple filter, is provided as a scaffolding within the platform). The MEPP2 GUI (based on Qt4/Qt5 library) allows to handle multiple objects (of different data structures) in one or more child windows and offers two types of visualization: the *Space* mode in which several objects are treated in the same viewer, and the *Time* mode in which several objects are seen as a sequence and can be visualized using a 3D+t video player. The platform offers an OpenGL accelerated rendering with Vertex Buffer Object (VBO) via the OpenSceneGraph 3.6 library. Several shaders are implemented including Blinn-Phong and Cook-Torrance with direct or indirect lighting. The platform supports PBR texture maps and face/vertex colors (see Fig. 2).

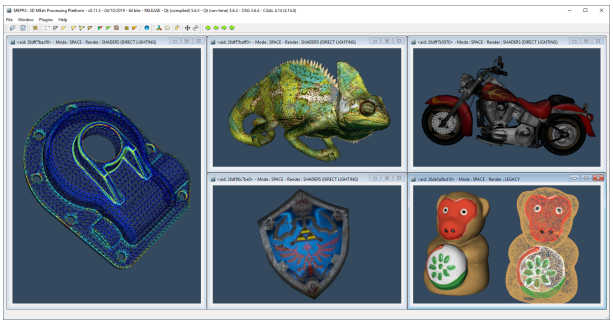

Figure 2: *Illustration of the MEPP2 graphical interface.*

3. Sample code

This section illustrates examples of generic and structure-specific source codes (C++11).

Structure-specific code example

```
typedef CGAL::Cartesian<double> K;
typedef CGAL::Surface_mesh<K::Point_3> SurfaceMesh;

void native_print_points(const SurfaceMesh &m)
{
  auto vb = m.vertices().begin();
  auto ve = m.vertices().end();
  for(; vb != ve; ++vb)  // loop over vertices
  {
    std::cout << m.point(*vb) << std::endl; // access geometry
  }
}
```

The above code, that illustrates how to iterate on the vertices of a mesh to print their coordinates, is specific to the CGAL Surface Mesh data structure. Applying such a processing to another data structure requires its rewriting.

Generic code example

```
template< typename VertexListGraph >
void generic_print_points(const VertexListGraph &g)
{
  // create a geometry object to manipulate the geometry
  FEVV::Geometry_traits< VertexListGraph > gt(g);

  auto pm = get(boost::vertex_point, g); // retrieve Point map
  auto vb = vertices(g).first;
  auto ve = vertices(g).second;
  for(; vb != ve; ++vb) // loop over vertices
  {
    auto p = get(pm, *vb); // access geometry
    std::cout << gt.get_x(p) << ","
              << gt.get_y(p) << ","
              << gt.get_z(p) << std::endl;
  }
}
```

The above code, based on the FEVV template library, is the generic equivalent of the previous code. It works with all data structures supported by MEPP2 (OpenMesh, CGAL's structures, AIF, PCL).

4. Available filters and plugins

This section describes the filters currently available in the platform. A filter is a program that either transforms or computes information on meshes or point clouds. For each filter, we indicate if it is coded in a *generic*, or *structure-specific* way and if it is integrated as a *plugin* in the graphical user interface.

Generic reader and writer (*Generic*): these reader and writer can load manifold and non-manifold mesh data structures (complex edges are duplicated for manifold data structures), and can handle common vertex/edge/face attributes via the FEVV property map concept. Readable/writable mesh formats include OBJ, OFF, PLY, and VTK. Readable/writable point cloud formats are OFF, PLY, XYZ, and PCD.

Curvature computation (*Generic*; *Plugin*): the algorithm from Cohen-Steiner and Morvan [CSM03] is implemented. Two versions are provided, the first considers a 1-ring neighbourhood around each vertex and the second acts on a geodesic neighbourhood according to a given radius of integration. Each version outputs minimum and maximum curvature values and directions.

Boolean operations (*Generic*; *Plugin*): a fast Boolean operation algorithm [LBD10] between 3D meshes is implemented; this algorithm can compute the union, intersection, and difference between two 3D meshes.

Perceptual quality metric (*CGAL-specific*; *Plugin*): the MSDM2 perceptual metric [Lav11] is implemented. Given a distorted 3D shape and a reference one, it computes a quality score that predicts the perceived distortion between them, as well as a per-vertex distortion map.

Just noticeable distortion model (*Generic*; *Plugin*): our platform integrates the just noticeable distortion (JND) model recently proposed by Nader et al. [NWFD16]. This perceptually-driven model integrates mathematical models of contrast sensitivity function and contrast masking. It was calibrated using an experimental study of the properties of the human visual system. It can predict, for a given illumination and screen resolution, whether a change in local contrast on a 3D mesh, induced by a local geometric distortion, is visible or not.

Progressive compression (*Generic*; *Plugin*): the progressive compression algorithm from Lee et al. [LLD12] is implemented. This algorithm can compress manifold 3D meshes (optionally with vertex colors) with high compression ratio and in a way that allows progressive decoding.

5. Applications

5.1. Comparing data structures

As an illustration of the usefulness of implementing algorithms generically, we ran the progressive compression and decompres-

sion filters on several 3D meshes, using four different data structures: CGAL Polyhedral Surface, CGAL Surface Mesh, CGAL Linear Cell Complex, and OpenMesh. The same generic code was used with all data structures. We compared the computation times and the memory footprints obtained for four meshes of growing sizes from 9000 vertices to 550000 vertices. The results presented in Figure 3 indicate that when considering the compression computation time criterion, OpenMesh is the fastest data structure followed by CGAL Surface Mesh which is 1.3 times slower. CGAL Surface Mesh provides the smallest memory footprint. Detailed results are available in the supplementary material.

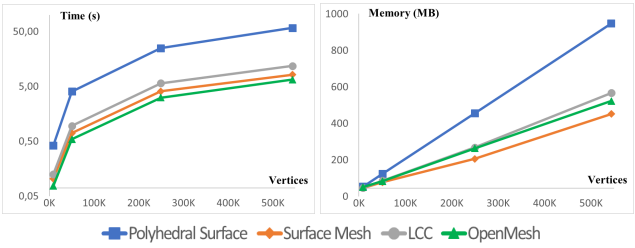

Figure 3: *Comparison of data structures (timing in log scale and memory in linear scale) for the compression of 4 different meshes.*

5.2. Complex processing pipelines

MEPP2 facilitates the development of new geometry processing filters and applications. The code below illustrates a complex processing pipeline using several data structures (the full source code is available in the supplementary material). It loads a point cloud in a PCL data structure, computes the point normals with PCL, then reconstructs a mesh from the point cloud with CGAL, and finally compresses the mesh with the progressive compression algorithm from MEPP2.

```
// load point cloud with FEVV generic reader
FEVV::PCLPointCloud  pc;
FEVV::PMapsContainer pmaps_bag; // FEVV bag of property maps
FEVV::Filters::read_mesh(input_file, pc, pmaps_bag);

// compute normals with PCL
[...]
pcl::search::KdTree<...>::Ptr kdtree(new pcl::search::KdTree<..>);
pcl::NormalEstimation<...> normal_estimator;
normal_estimator.setInputCloud(pc.makeShared());
normal_estimator.setSearchMethod(kdtree);
normal_estimator.setKSearch(18);
normal_estimator.compute(pc);

// copy PCL data structure into CGAL-compliant data structure
[...]
std::vector<...> points;
for(auto p : pc)
  points.push_back(...(Point(p.x, p.y, p.z),
                       Vector(p.normal_x,p.normal_y,p.normal_z)));

// reconstruct mesh with CGAL (Poisson method)
[...]
CGAL::Polyhedron_3<...> reconstructed_mesh;
CGAL::poisson_surface_reconstruction_delaunay(
      points.begin(), points.end(),
      ..., reconstructed_mesh, ...);

// compress mesh with FEVV Compression Valence filter
[...]
auto pm = get(boost::vertex_point, reconstructed_mesh);
FEVV::Filters::compression_valence(reconstructed_mesh,
        &pm, ..., "cloud_to_mesh.compressed.p3d", ...);
```

6. Conclusion

In this paper, we presented the open-source MEPP2 platform, which provides both an API for processing 3D meshes and point clouds and a graphical user interface (GUI). The strength of the platform is its core layer, the FEVV template library, that relies on a set of concepts that provide an abstraction layer over several third party mesh and point cloud data structures. We illustrated the benefits of this architecture with several applications. As the abstraction layers rely on templated and inlined functions that wrap data structure function calls, we expect to encounter only a small overhead. Nevertheless, the use of associative property maps for non-indexed data structures can lead to a performance loss. We plan to support the volume mesh representation through the introduction of new concepts. Our main objectives are to ease the processing of 3D data (point cloud, surface, volume, their attributes) in an interoperable way, and to federate a user community.

References

[BSBK02] BOTSCH M., STEINBERG S., BISCHOFF S., KOBBELT L.: OpenMesh - a generic and efficient polygon mesh data structure. *OpenSG Symposium* (2002). 1

[CCC08] CIGNONI P., CALLIERI M., CORSINI M.: Meshlab: an open-source mesh processing tool. In *Eurographics Italian Chapter Conference* (2008). 1

[CSM03] COHEN-STEINER D., MORVAN J.: Restricted delaunay triangulations and normal cycle. In *19th Annu. ACM Sympos. Comput. Geom.* (2003). 3

[Gir19] CloudCompare (version 2.10) [GPL software], 2019. Retrieved from http://www.cloudcompare.org/. 1

[GJS*06] GREGOR D., JÄRVI J., SIEK J., STROUSTRUP B., DOS REIS G., LUMSDAINE A.: Concepts: Linguistic support for generic programming in C++. In *ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications* (2006), pp. 291–310. 2

[JP*18] JACOBSON A., PANOZZO D., ET AL.: libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/. 1

[Lav11] LAVOUÉ G.: A Multiscale Metric for 3D Mesh Visual Quality Assessment. *Computer Graphics Forum 30*, 5 (2011), 1427–1437. 3

[LBD10] LECONTE C., BARKI H., DUPONT F.: *Exact and Efficient Booleans for Polyhedra*. Tech. rep., 2010. URL: http://liris.cnrs.fr/Documents/Liris-4883.pdf. 3

[LLD12] LEE H., LAVOUÉ G., DUPONT F.: Rate-distortion optimization for progressive compression of 3D mesh with color attributes. *The Visual Computer 28*, 2 (may 2012), 137–153. 3

[LTD12] LAVOUÉ G., TOLA M., DUPONT F.: MEPP - 3D Mesh Processing Platform. In *International Conference on Computer Graphics Theory and Applications (GRAPP)* (2012). 1

[NWFD16] NADER G., WANG K., FRANCK H., DUPONT F.: Just Noticeable Distortion Profile for Flat-Shaded 3D Mesh Surfaces. *IEEE Trans. on Visualization and Computer Graphics* (2016). 3

[RC11] RUSU R. B., COUSINS S.: 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation* (2011). 1

[SG03] SILVA F., GOMES A.: Adjacency and incidence framework - A data structure for efficient and fast management of multiresolution meshes. In *GRAPHITE* (2003), pp. 159–166. 2

[The20] THE CGAL PROJECT: *CGAL User and Reference Manual*, 5.0.1 ed. CGAL Editorial Board, 2020. URL: https://doc.cgal.org/5.0.1/Manual/packages.html. 1