A Sheaf System Primer

David M. Butler

Limit Point Systems, Inc.

In this document we give a high level overview of the Limit Point Systems, Inc, (LPS) Sheaf System by answering two questions: What is it? and What is it good for? We then amplify the answers with a short example that introduces some of the terminology, concepts, and methods of the system.

1 What is the Sheaf System?

The LPS Sheaf System is:

- A collection of carefully chosen mathematical concepts, structures, and procedures. These mathematical techniques collectively provide a terminology for describing real world problem domains, especially problem domains involving physical science and engineering disciplines. These techniques are particularly well suited to "joined-up" (integrated) science in problem domains featuring complex multi-level structures with multiple property fields using different numerical representations. Geoscience is a prototypical example.

- A data model based on the collection of mathematical concepts that captures conceptual aspects, computational aspects, and persistent data aspects of a problem domain.

- A suite of software frameworks, classes, and tools that facilitate direct implementation of new applications, or encapsulation of legacy applications, as described using the mathematical terminology.

2 What is the Sheaf System good for?

The Sheaf System can potentially make significant contributions to resolving a number of recognized information technology issues in computational science. Some examples include:

- Interoperation of diverse geometry and property formats: the Sheaf description of geometry and property data implies a mathematically general and application independent interface (the "Sheaf API") for accessing and manipulating the data. This API is implemented in the Sheaf class libraries. An adapter can be implemented for each legacy format that provides the operations of the Sheaf API on top of the legacy format. The legacy data can then be accessed and manipulated through the adapter, by invoking the Sheaf operations exported by the adapter. This means that algorithms for translating and otherwise interoperating diverse formats can then be written using only the operations of the Sheaf API, without referring directly to the operations of the legacy data. Hence these algorithms are format independent. As a result, if we want to support interoperation of N formats, we need to develop and maintain N Sheaf adapters instead of N*(N-1) direct format-to-format translators. We provide a more detailed example of such an application independent translation service in the next section. An additional benefit of this approach is that even without implementing the adapter for a specific legacy system, the specification of the Sheaf adapter API can be used to check for functionality gaps in a legacy system: any API call the legacy system cannot support points to a specific gap.

- Visualization of diverse geometry and property formats: the Sheaf description and API support the implementation of format independent visualization algorithms in a manner analogous to that just described for data translation. To support M visualization techniques for N formats, we need to develop only M format independent visualization routines, not N*M format specific visualization routines.
- Effective use of parallel computing: the Sheaf System concepts facilitate identification and description of the parallelism inherent in a problem domain at the conceptual level and provide for the propagation and refinement of that information into the computational and persistent data aspects. The concepts and their embodiment in the Sheaf class libraries facilitate platform independent parallel implementations of the data manipulation and visualization algorithms described above and potentially enable implementation of application and platform independent, automatic problem decomposition and distribution services.
- Long-term, archival storage of technical data: the Sheaf System provides a mathematically precise, and provably general method for schematizing technical data. Since the method is derived from the mathematical semantics of the problem domain, it does not depend on persistence of the application that generated the data. Data storage using such schema can be implemented in a variety of ways that can evolve with the technology environment. The current Sheaf System provides one implementation based on the HDF5 file system, a file format widely used in the computational science community.
- Formal language and/or ontology: the Sheaf concepts can potentially be developed into a formal data definition and manipulation language. Such a "SheafQL" could play a role similar to the role SQL has played in increasing programmer productivity and facilitating application integration in business information systems. Similarly, the Sheaf concepts can potentially be developed into a formal ontology for Semantic Web applications. Both the formal language and the ontology would inherit the natural expression of parallelism provided by the Sheaf concepts.

3 An example: re-sampling a material property field

We illustrate the key Sheaf concepts through a somewhat detailed example, introducing some of the terminology as we go along. We hope that the context of the example will convey enough of the meaning for the reader to get a general feel for the Sheaf approach.

The example we'll describe is an application independent algorithm for translating a property field from one mesh representation to another. In the Sheaf terminology, we refer to this operation as "pushing" a property "section" from one mesh or "base space" to another. The Sheaf library contains a general implementation of the push operation.

The example is shown in figure 1. We have a triangle mesh, admittedly a very *small* triangle mesh so we can draw all the pictures easily, with a single scalar property, density. The set of all possible property values forms a space which we refer to as the "fiber space" for the property. The association between points in the base space and values in the property fiber space is defined by a map, or "property section", indicated by an arrow in the diagram.

We consider the base space to be an abstract shapeless "point set" or "topological space" and treat its actual shape and geometry as a special kind of property. The fiber space for the geometry property is E2, ordinary two dimensional Euclidean space. Just as with density, the value associated with each abstract point, that is, the geometric position of the point, is defined by a map, the "coordinate section". Geometry is a special kind of property because the coordinate section must be invertible, indicated by the double-headed arrow in the diagram. Note that although treating position the same way we treat properties originates in very fundamental mathematical principles, it also just formalizes common computing practice: both coordinates and properties are typically stored in arrays sharing a common indexing pattern.

Also shown in the diagram is a second mesh, even simpler than the first: a line with just two segments. The line base space is also treated as abstract and its geometry specified by an invertible coordinate section. The line mesh does not have the density property defined on it yet and the objective of this example is to describe how we can compute a property section for the line mesh.

Naively, we want the line mesh property to be "the same" as the property on the triangle mesh and we have to consider both conceptual aspects and computational aspects to specify what this means. We'll deal with the conceptual aspects first, then discuss the computational accuracy issues once we've described the basic algorithm.

The conceptual aspects address the abstract mathematical definition of two sections being "the same". First notice that the property section associates a property value with each abstract point in the base space and the coordinate section associates a geometric position with each point. Because the geometry section is invertible, we can associate a property value with a geometric position by starting at the position, following the inverse coordinate section to a point in the base space, then following the property section to a property value. This chain defines property as a function of geometric position, instead of property as a function of abstract point.

The line mesh and the triangle mesh overlap in the geometry fiber space, as shown in the diagram. So conceptually, if we had "the same" property defined as a section on the line mesh, then for positions in the overlap we could form property as a function of position in two ways:

1. We could go from the position through the inverse coordinate section for the line mesh and then through the property section for the line mesh, or
2. we could go from the position through the inverse coordinate section for the triangle mesh and then through the property section for the triangle mesh.

If the line mesh property section is "the same" as the triangle mesh property section, then these two paths should yield the same property value, as least in the abstract.

We use this process of following sections to *define* the value of the property section on the line mesh. As shown in figure 2, we define the value of the line mesh property at a point to be the value we get by (1) following the line mesh coordinate section to a geometric position, then (2) following the inverse coordinate section for the triangle mesh, and finally (3) following the triangle mesh property section. The value found in step (3) is defined to be the value of the line mesh property section at the given point.

To describe how we implement this section-following process in a mesh independent manner, we have to return to a discussion of the base spaces. As we said above, the sheaf interpretation of a base space is that it is an abstract point set. This point set includes not just the vertices of the mesh, but, for the triangle mesh, all the implicit points in the interior and on the boundary of the triangles and, for the line mesh, all the points in the interior of the segments (the vertices contain the boundary for the line mesh). The triangles, segments, and vertices are treated as parts of their respective base spaces. Each whole mesh is itself considered a part and all the parts of each mesh are organized into a partially ordered set, or "poset" as shown in figure 3(a) for the triangle mesh and 3(b) for the line mesh. The visual representation of a poset is a graph, or "Hasse diagram", in which each part is a node in the graph and there is a link between two parts whenever one part "covers" or immediately includes another part. A path in the graph corresponds to inclusion. So in figure 3(a) the triangle mesh part covers triangle 0 (t0) and triangle 0 covers vertex 0 (v0). The triangle mesh part includes vertex 0 but does not directly cover it.

The graph is structured so that if part A covers part B, part B is below A on the page. So all the parts that are included in a given part can be found by following links downward from the given part. The subgraph formed in this way, by going down from a part, is called the down set of the part. Since any part includes itself, the down set of a part always includes the part itself. For instance, the down set of t0 is t0, v0, v1, and v3.

The representation of a section, that is, how the value of the section at each point in the base space is represented, can also be described a poset. To simplify the presentation, we can describe the section representation poset by annotating the base space graph with two kinds of information. First, we need to know what members of the graph the section evaluators are associated with, shown in the figures as an "E" tag. Second, we need to know what members of the graph the section data is associated with, shown in the figures as a "D" tag.

A section evaluator provides two functions:

- Value_at_coordinate takes as input a specification of which point in the evaluator part the section should be evaluated at and, using the data contained in the down set of the evaluator part, computes the value of the section at that point. Typically, the point specification is in the form of local coordinates, but other specifications are possible. The section data can be any kind of data and value_at_coordinate can be any computational procedure; but the most common case is that the data is the value of the section at some points and the computational procedure is interpolation. For instance, in our triangle mesh the section data is the value of the property or geometric position at the vertices and the computational procedure is linear interpolation.
- Coordinate_at_value is the inverse of value_at_coordinate. It takes as input a section value and returns as output a specification of the point at which the section has the given value. If such a point can't be found, it returns an error.

We have demonstrated it only for a very simple example but the poset representation of a section is general. We can show mathematically that any mesh, any decomposition of space, and any section representation based on decomposition of space and stored data can be represented as posets. That includes global function methods, which correspond to the simplest decomposition of all, just a single part - the whole space. We won't discuss it further, but we note that it is this poset representation that forms the basis for the mathematical schematization for long term archiving mentioned above.

Now we return to our push algorithm. In order to define the property section everywhere in the line mesh base space, we need the value of the section at the parts with the data tags. We can compute these values as follows:

Traverse the line mesh graph in depth first order. Each time we find a data tag, we will be in the down set of some evaluator, referred to here as the current evaluator. For each data part:

1. Get the geometric position for the data part by invoking the line mesh coordinates section value_at_coordinate function of the current evaluator part. We'll call this position the line mesh data position, as shown in figure 2 for data part v2.
2. Search the triangle-mesh graph for the evaluator part with a geometric region that contains the data position, triangle 1 in figure 2. This search can be done efficiently using spatial hashing or spatial trees.
3. Get the local coordinates in the triangle mesh evaluator part of the line mesh data position using the coordinate section coordinate_at_value function of the evaluator part.

4. Feed the local coordinates to the property section value_at_coordinate function of the evaluator part to get the property value. Store the property value in the data tag of the line-mesh data part.

When this algorithm completes, all the data tags in the line mesh have a property value and the line mesh property section is fully defined.

We've deliberately made this example as simple as possible, but we can now briefly sketch several extensions to more complex and realistic problems.

First, the poset description and the algorithm easily incorporate multiple kinds of interpolators. For instance, figure 3(b) shows the poset description for linear interpolation on the line mesh, while figure 3(c) shows the poset for constant interpolation.

Second, the poset description and algorithm easily incorporate arbitrarily complex structures. For instance, figure 4 shows the result of pushing a scalar property from a structure containing multiple layers and a salt bag, with different section representations on each part, to a regular ijk grid.

Third, domain decomposition for parallel computing also fits easily into the poset description and algorithm, so the push algorithm is not only general, but can also be fast.

Finally, we return to the issue of computational accuracy. The discussion above ignored the fact that in general the interpolation provided by the target, the line mesh in this case, will not exactly match the interpolation provided by the source, the triangle mesh, and hence the result of the push cannot be exactly "the same" as the source. Instead, the target approximates the source and in practice it is important to be able to ensure that the accuracy of the approximation satisfies some given criterion. The algorithm can be extended to automatically refine the target mesh to achieve some specified accuracy criterion. This "refining section pusher" algorithm is also implemented in the Sheaf library, as a class template with one of the template parameters being the accuracy criterion or "refinement policy". The refining pusher can be instantiated with any one of several such policies provided by the library, or a custom policy provided by the client. Figure 5 shows the result of a refining push in a paleo-geology application.

The basic algorithm described above and these extensions can be implemented entirely in terms of the Sheaf API, without reference to mesh format specific features, and hence will operate on any mesh format equipped with a Sheaf API adapter. Thus, although we need to develop a Sheaf API adapter for each format, the translation or push algorithm need only be implemented once, not once for each pair of formats. In fact, as mentioned above, the push algorithm already *has* been implemented.

4 Summary

The LPS Sheaf System is a collection of mathematical tools; a data model; and a software suite that can make important contributions to resolving recognized IT issues in computational science. The push operation between two simple meshes serves to illustrate some of the concepts; hopefully it suggests how the system works and how it can be used.

5 Acknowledgements

Mark Verschuren, Philipp Konerding and GameChanger, all at Shell International Exploration and Production, Inc., and Christoph Ramshorn, Petroleum Data Systems, provided many useful comments and editorial suggestions.

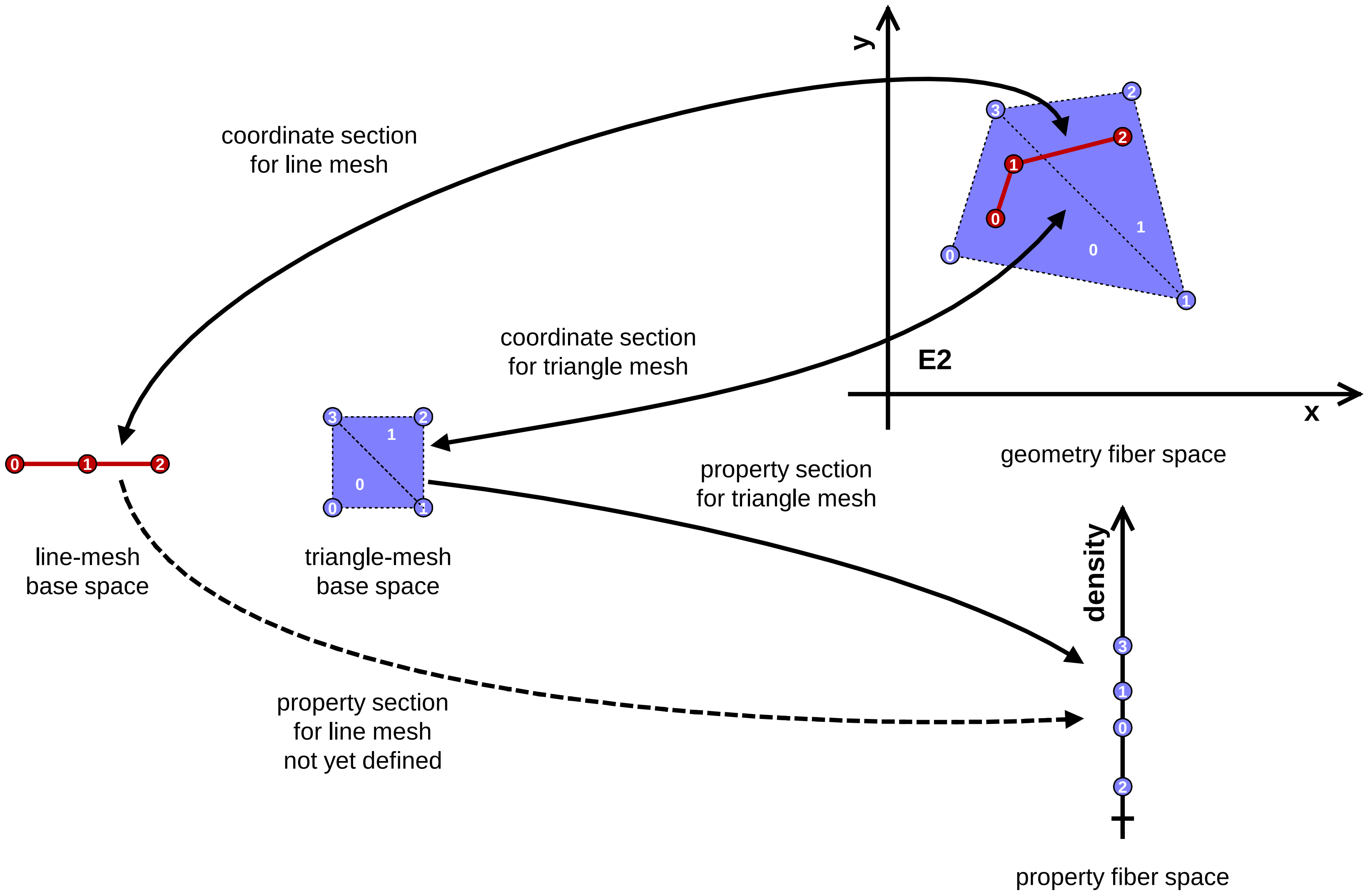

y
coordinate section for line mesh
coordinate section for triangle mesh
E2
x
geometry fiber space
property section for triangle mesh
density
line-mesh base space
triangle-mesh base space
property section for line mesh not yet defined
property fiber space

Figure 1: Section push example

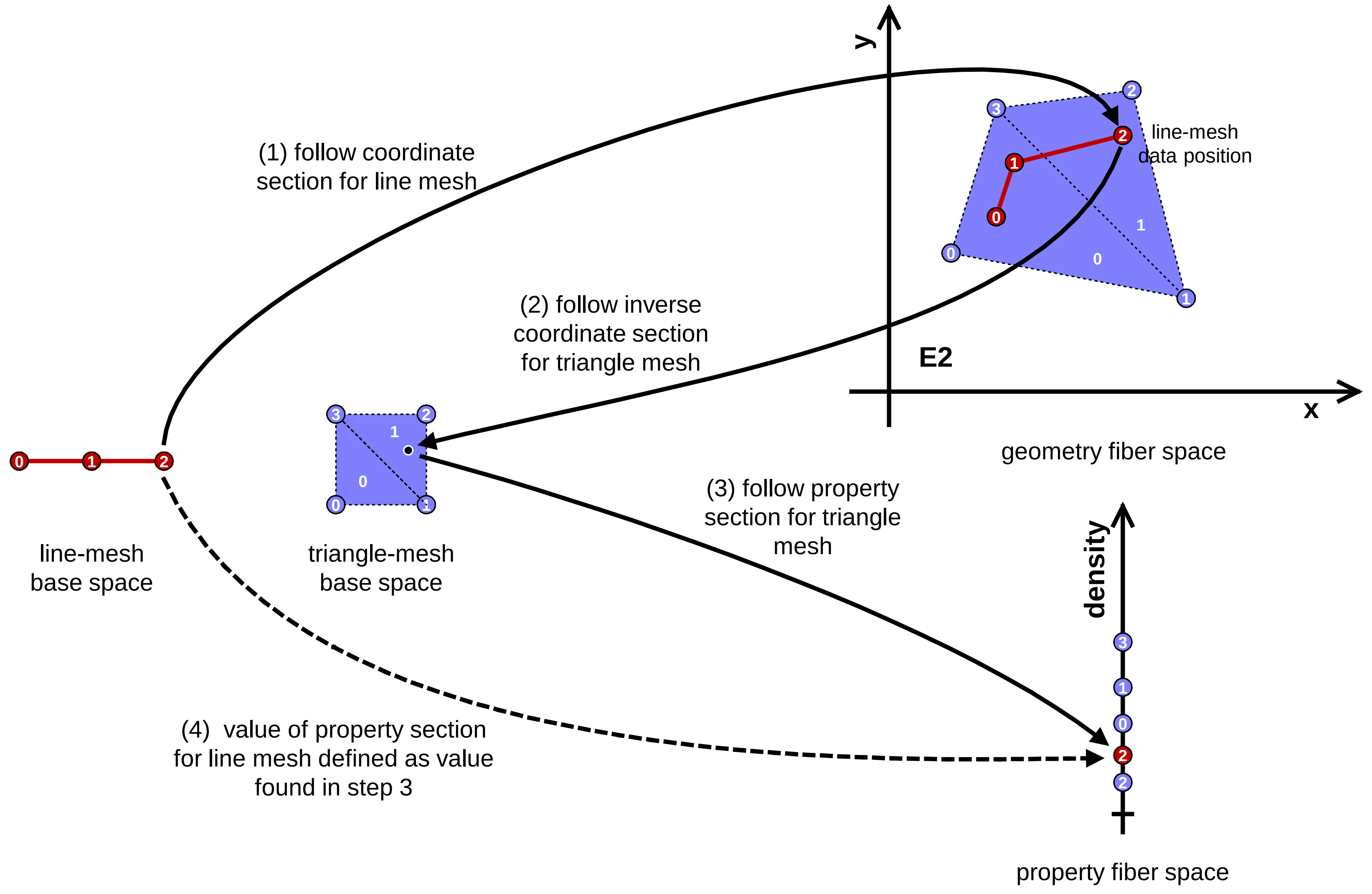

y
(1) follow coordinate section for line mesh
line-mesh data position
(2) follow inverse coordinate section for triangle mesh
E2
x
geometry fiber space
line-mesh base space
triangle-mesh base space
(3) follow property section for triangle mesh
density
(4) value of property section for line mesh defined as value found in step 3
property fiber space

Figure 2: Property value definition for push operation.
The line mesh property section (dotted arrow) is defined by following the other arrows in the order shown.

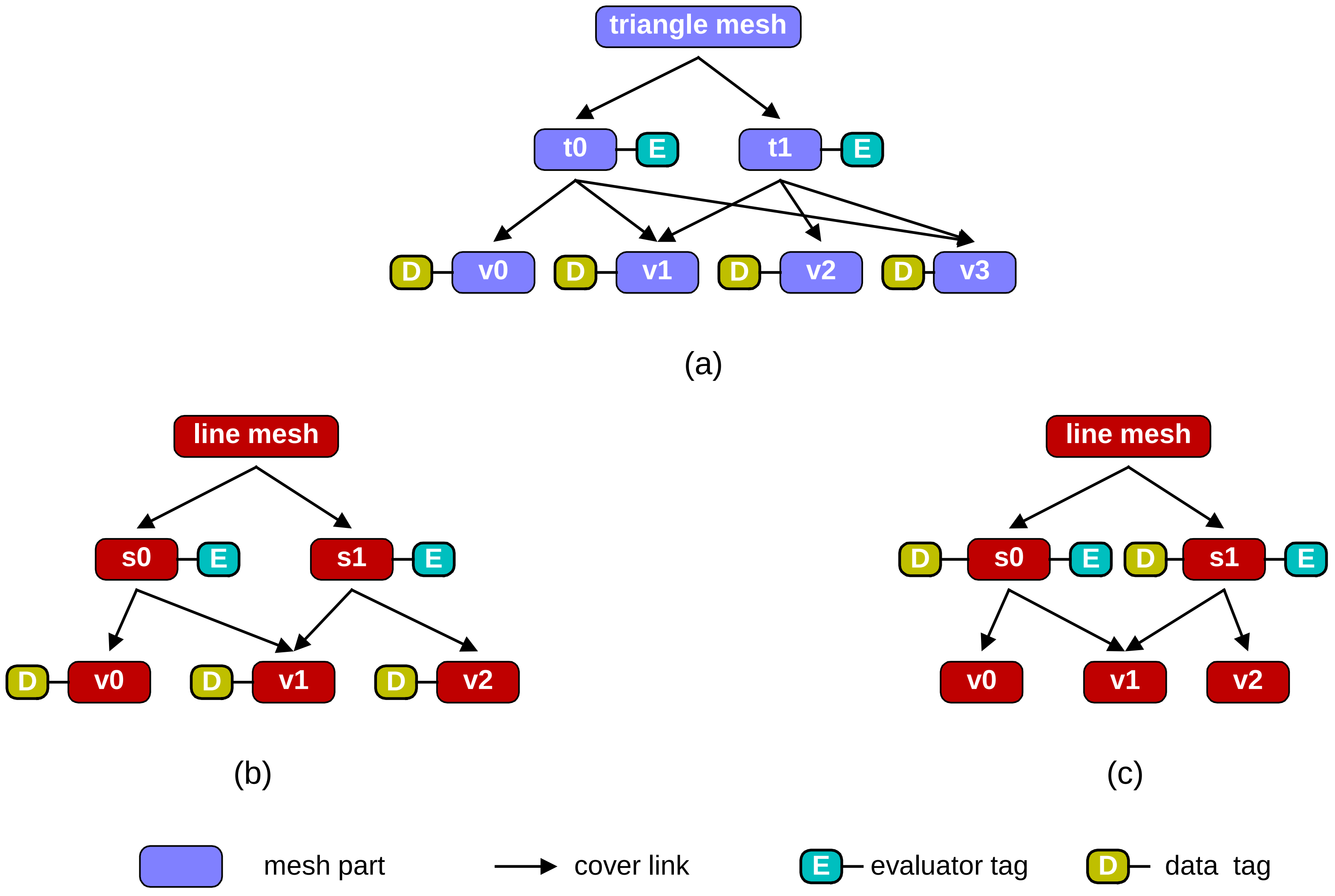
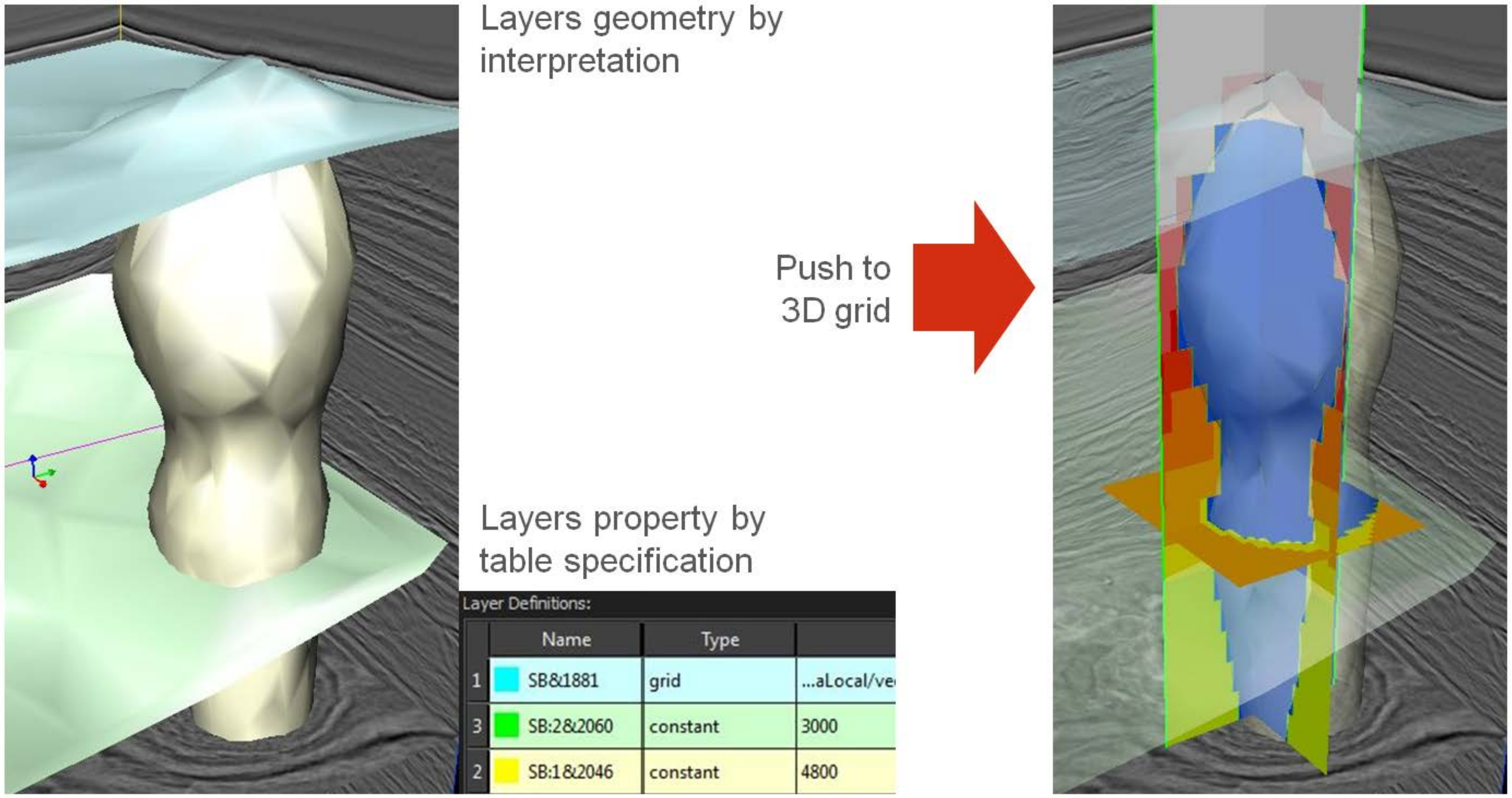


Figure 3: Tagged posets


Layers geometry by interpretation
Push to 3D grid
Layers property by table specification

Layer Definitions:

	Name	Type	
1	🟦 SB&1881	grid	...aLocal/ve
3	🟩 SB:2&2060	constant	3000
2	🟨 SB:1&2046	constant	4800

Figure 4: Push of a scalar property from a complex structure with multiple layers, a salt bag, and different property representations on each part to a regular ijk grid. The source of the push is shown on the left, where the predominantly horizontal surfaces represent the boundaries between the layers. The target is shown on the right, where the scalar property is represented by color on 3 axis-aligned cutting planes. Courtesy of Shell International E&P.

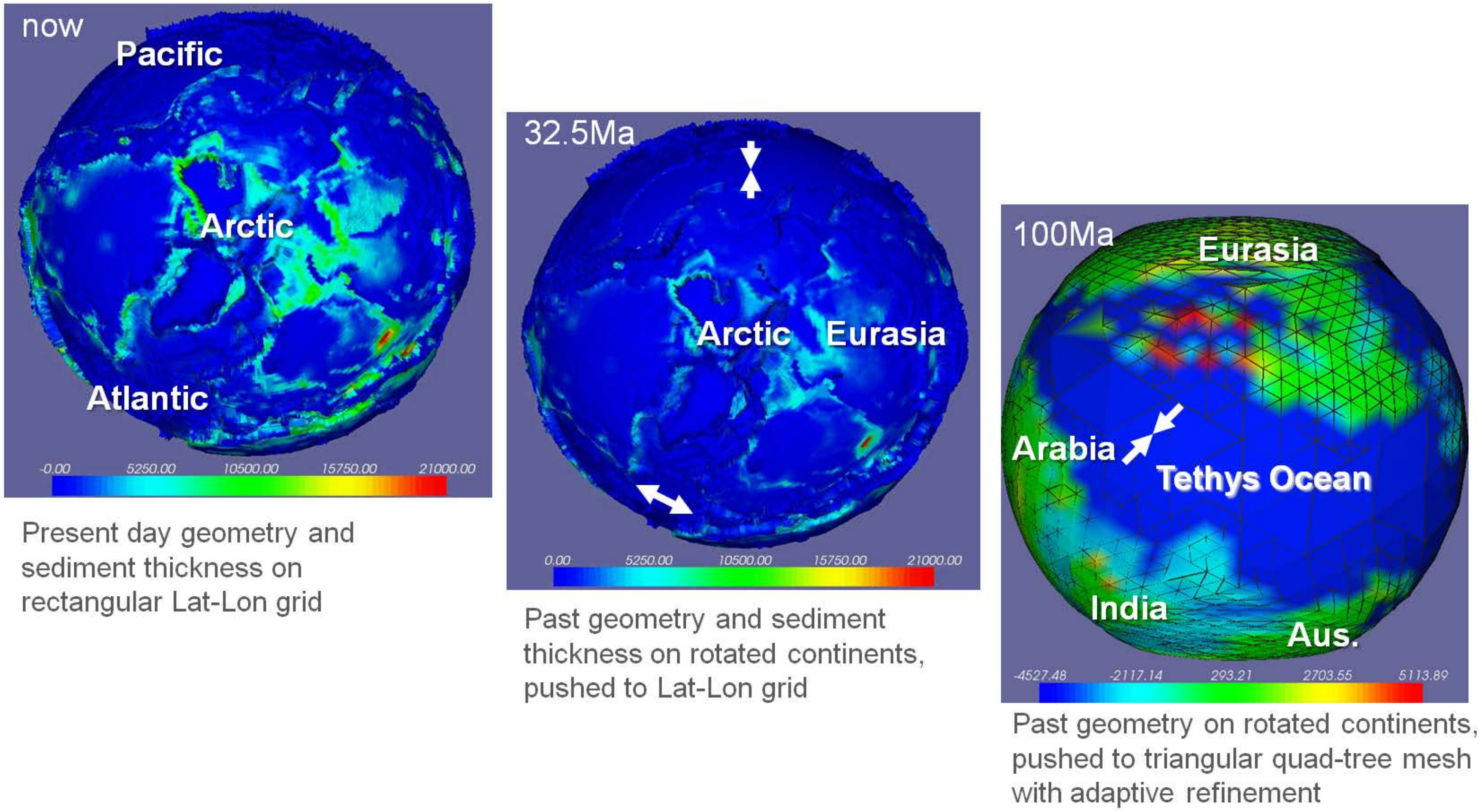

now
Pacific
Arctic
Atlantic
-0.00
5250.00
10500.00
15750.00
21000.00
Present day geometry and sediment thickness on rectangular Lat-Lon grid
32.5Ma
Arctic
Eurasia
0.00
5250.00
10500.00
15750.00
21000.00
Past geometry and sediment thickness on rotated continents, pushed to Lat-Lon grid
100Ma
Eurasia
Arabia
Tethys Ocean
India
Aus.
-4527.48
-2117.14
293.21
2703.55
5113.89
Past geometry on rotated continents, pushed to triangular quad-tree mesh with adaptive refinement

Figure 5: Push of present day geometry and sediment thickness to paleo-times without and with adaptive refinement. Courtesy of Shell International E&P.