Programming Topological Operations for Visual Illustrations in an Introductory Geomorphology Course

R. Bezin[1], B. Crespin[1], X.Skapin[2], P. Meseure[2] and O. Terraz[1]

[1]XLIM/DMI - UMR CNRS 7252, Université de Limoges, France
[2]XLIM/SIC - UMR CNRS 7252, Université de Poitiers, France

Abstract

In the context of teaching geomorphology phenomena, producing illustrations and animations can be a tedious process. We propose an experimental framework, dedicated to 3D erosion and sedimentation modeling written in C++, combined with an existing topological modeler. Using the "generalized maps" as the underlying 3D model, we process each case of collision between elements in the scene in order to guarantee both topological and geometrical coherence during user-defined animations. Erosion and sedimentation operations can be combined to manipulate evolution scenarios leading for example to the creation of arches, bridges, tunnels or caves. Some of these scenarios, implemented in our framework with the help of a geology teacher, are presented in this paper in order to show the technical feasibility of our project before developing new ones.

Categories and Subject Descriptors (according to ACM CCS): K.3.1 [COMPUTERS AND EDUCATION]: Computer Uses in Education—Computer-assisted instruction (CAI) I.3.7 [COMPUTER GRAPHICS]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Although the landscapes surrounding us may seem static, they are continually evolving at a variety of time and spatial scales. These phenomena are mainly studied by *geomorphology*, which seeks to explain the history and dynamics of landforms. To produce visual illustrations of such phenomena to teach introductory geomorphology can be a tedious process: most of them come as a set of successive images, usually in 2D as shown of Figure 1 (top) or in pseudo 3D (bottom). These illustrations are usually created thanks to dedicated software such as Geographic Information Systems (GIS) or traditional 3D modeling tools: Blender, Maya, etc.

In this paper we chose to focus on *erosion and sedimentation phenomena*, which can lead to the formation of complex geometries such as natural arches, bridges or tunnels. In this case, relying on existing software is not possible since 3D illustrations are mandatory and complex *topological* events can occur that usually require expensive volumetric representations.

Our framework is able to simulate the geomorphological evolution of a 3D terrain represented as a set of volumes, relying on a topological model named "generalized maps", along with atomic operations to handle topological events in a robust way. These operations can be overridden to implement complex evolution scenarios in a modeling software based on generalized maps, by defining C++ functions that process collisions occurring between vertices, edges or faces. This paper describes different early experiments with a geomorphology teacher to define such C++ code and produce visual illustrations of erosion and sedimentation phenomena that will be used in an introductory geology course.

Our work addresses various topics of CG-based education: first of all, we want to create an educational program combining topological representation with geomorphology. We also intend to show how rather simple C++ code can produce visual illustrations for teaching purposes. A primary objective for these illustrations is to enhance student learning and help understand complex geomorphological phenomena.

This paper is organized as follows: the next section presents different works related to visual and programming tools in geomorphology and other disciplines, and gives an insight for the reader to understand the basic concepts of

DOI: 10.2312/eged.20141025

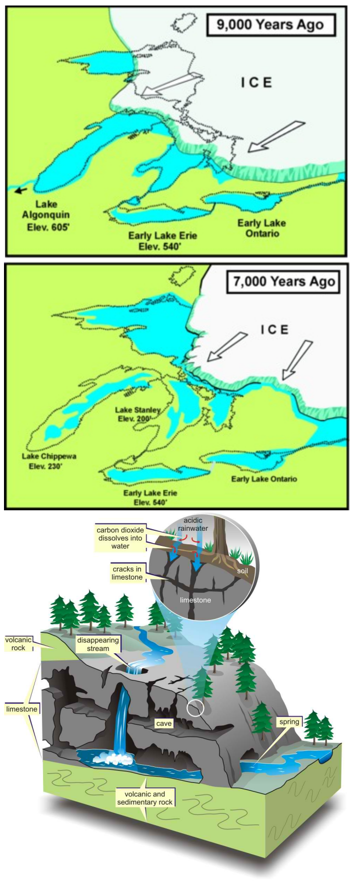


Figure 1: *Examples of visual illustrations in 2D and pseudo 3D commonly found in introductory geomorphology classes*

topological modeling. Section 3 describes our graphical and programming environment, and shows how our main simulation loop can be modified to adapt to various scenarios. In Section 4 we present different experiments conducted with a geologist in order to produce simple animations of erosion and sedimentation phenomena, which are discussed in the final section.

2. Related Works

2.1. Education papers

The goal of our work is to produce visual illustrations of complex phenomena in Earth science; of course it has already been shown that graphics, images and animations are able to enhance learning in many disciplines [MB05].

Producing such illustrations for a geology or geomorphology course could rely on advanced terrain generation methods such as [PGMG09], however none of these works address the specific problem of topological changes in the 3D model. Other methods are specifically targeted towards geology, sometimes based on high-level user interactions. A graphical system that uses geological sketches is described in [LHV12]. Their solution is built on a flip-over metaphor that sketches the individual steps of a "geological story". In [SVBCS13] the system can be used to visually explore and annotate geological outcrops through multitouch interactions, including a 3D navigation technique and horizon surface creation and edition. But again these softwares are more oriented towards the creation of geological representations than their evolution. Another problem is the human resources and task force necessary to develop such advanced programs which are usually restricted to specific API and hardware. It could be possible to rely on computer graphics students to develop advanced software as shown in [Lie13]: these students can benefit from having software engineering projects in other departments, but in our opinion this approach still requires a significant amount of help from CG teachers and researchers to develop and maintain such software.

Another way to address these problems is to consider the help of scripting or programming. Programming is now a wide-spread task in the scientific community, especially in geology where GIS software is rather common [PJR*11]. Bridging the gap between scientific researchers and programming tools developers through the practice of computational science is actually a rather old idea [FP75], and introductory computer science courses are now wide spread in science curricula.

Therefore, a non-visual software that relies on an API in C++ or another programming language can now be used by non-programmers, and developing such tools is certainly less time-consuming and much more stable. Numerous examples can be found, such as *Processing* [RF06], where programming is used by students, artists, designers, architects, and researchers to learn, prototype and produce high-level animations. Of course, there is no shortcut in learning to program and sufficient time to become familiar with basic concepts is needed [HM06].

Finally, it should be noted that simplified APIs aimed at non-programmers can be developed to "hide" most of the difficult tasks, as for example in [SCN12] which describes a simple parallel computing framework intended to avoid developing with CUDA for different scientific applications. In this case again we believe that an important effort is needed from computer scientists to design and maintain such APIs but this approach could be a future direction for our work.

2.2. Topological structures

Our topological model is based on *generalized maps*, an extension of combinatorial maps that can represent the topological structure of a subdivided volume composed of vertices, edges, faces and volumes linked together by adjacency

/ incidence relationships. Its compact and uniform representation simplifies definitions and evolution algorithms in any dimension. It also makes it easier to implement mesh modifications such as triangulation, insertion and subdivision, as well as neighborhood search in constant time.

A n-dimensional generalized map (or n-G-map) is a set of abstract elements called *darts*, defined in an homogeneous way for any dimension n, linked by combinatorial involutions. A 3-G-map can be seen as a generalization of similar topological representations such as *winged edges* or *half edges* which use explicitly linked structures to store the geometry and topology of faces, edges, and vertices. This is illustrated in Fig. 2 with two views of the same two volumes sharing a common face. The bottom view shows a graphical representation of darts which define the topological structure.

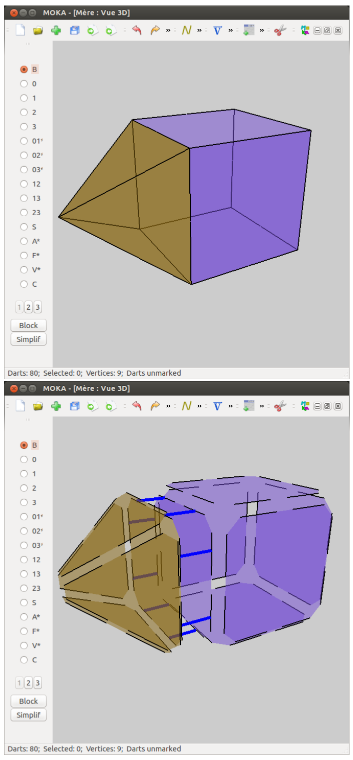


Figure 2: *Compact and exploded views of two volumes sharing a common face*

For our project it is sufficient to say that darts can represent vertices, edges or faces depending on the context of an erosion operation; interested readers can refer to [Lie94] for a complete presentation of n-G-maps.

3. Graphical and Programming Environment

3.1. Overview

Defining an erosion/sedimentation scenario in our system is basically a two-steps process:

- Use the modeler to define the initial topology and geometry of the volumetric elements and define how a selected set of vertices will move during the animation. In the first picture in Fig. 3 the selected vertex appears marked with a red cross, and its direction is set vertically downwards: it will be displaced when running the scenario as in the second picture.
- Define what should happen in case of *topological inconsistencies*, such as a moving vertex colliding with a volume, using C++ code. In our example this leads to the creation of new vertices (third picture, when the yellow vertex collides with the bottom volume) to maintain the consistency of the topology.

The main difficulty is to define how to create new vertices during the animation and how they should behave in order to obtain the desired result. Our API provides a set of C++ operations for this purpose such as `vertexFaceCollision(v,f,&newV)` which fills a list of new vertices `newV` resulting from the collision of a moving vertex `v` with a face `f`. Section 4 gives several examples where these operations are used to describe different erosion/sedimentation scenarios (including the basic digging operation shown here) within our framework.

An event queue stores all initial and new moving vertices during the animation, and a C++ class called `geo` represents the context of the whole operation.

3.2. Topological modeler

Our framework relies on Moka, a 3D geometrical modeler based on a 3-G-map kernel [VD], which contains different operations such as basic object creation, cell insertion, removal and contraction, and triangulations. Our topological operations run as an additional library that interacts both with the kernel and the modeler libraries.

The initial 3D model can be imported into Moka from an OBJ file created with another 3D modeling software, or defined from scratch by the user. Moka provides all necessary operations, and can also apply basic CSG operations such as removing a part of the model for inside visualization (see Fig. 3, bottom picture).

In our project, the set of moving vertices (associated with specific darts of the 3-G-map) must be selected by the user before the evolution scenario starts. They appear in red and are marked by a specific boolean value called `isMarked` in the C++ code. We will use the API provided by Moka and the ability to define any type of mark associated with darts to implement our own operations described in the remainder of this paper.

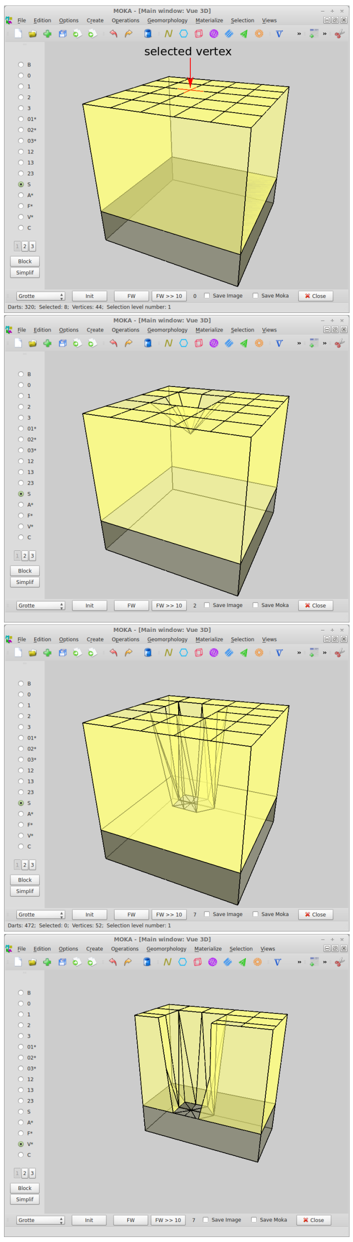


Figure 3: *Progressive steps of a simple erosion operation generated with our framework from a selected vertex going downwards. The last picture shows an inside view of the model obtained with a CSG difference.*

3.3. Evolution of topological structures

All vertices stored in the event queue move during the animation, and as stated previously the user must define how new vertices created during the animation should behave. The following structure describes a moving vertex:

```
typedef struct {
 CDart* dart; // position of the moving vertex
 CVertex dir; // direction
 float t; // time of collision
 CDart* obj; // dart (vertex, face or edge) colliding with
       the moving vertex
} Move;
```

New vertices are created only in case of specific events, *i.e.* collisions between a moving vertex and different darts (another vertex, an edge or a face). Fig. 4 summarizes the 9 possible cases that can occur. These collisions are not symmetric, since only one moving vertex M is processed at a given time. For example, a *vertex-face* collision means that M crosses a face F that it does not belong to (but M and F can belong to the same volume). In geology this configuration could correspond to a stalactite reaching the ground. A *face-vertex* collision implies that the face which M belongs to moves and collides with another vertex, for example if, on the contrary, the ground being uplifted by sedimentation reaches a static object.

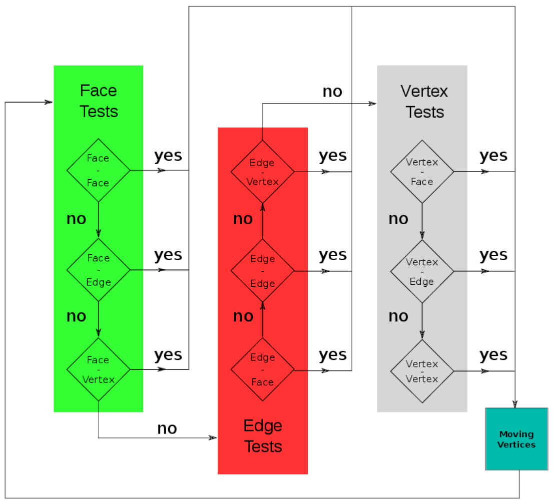


Figure 4: *9 different types of collisions that can occur in our model*

Each of these 9 cases triggers a default method defined by three main steps in C++:

- call a procedure from our library that may return a set of new vertices
- define the properties of these new vertices
- add these vertices in the main event queue if they are set to move at the next step

All 9 methods can be overridden for his/her own purpose by the user, who must also define an initialization function called `init` at the beginning of the animation.

4. Experimental results

Experiments with our framework were conducted with a geology professor who had basic notions of C++ programming. In the following we describe three different scenarios that can be combined together to obtain a rather complex animation.

4.1. Digging a hole with differential erosion

As shown on Fig. 3, a hole can be created by the vertical motion of a single vertex; troubles begin when this moving vertex collides with a volume of rock that is less sensitive to erosion (this phenomena is called *differential erosion*). The erosion rate can be set in Moka thanks to a user-defined field value. First let us have a look at the `init` function:

```
void ScenarioDigging::init() {
 // define a vertical direction
 CVertex dirErosion = CVertex(0,-0.25,0);
 int markV = ctrl->getSelectionMark();
 // find selected vertices
 for(CDynamicCoverageAll it(map); it.cont(); ++it) {
  if(map->isMarked(*it, markV)) {
   Geomorphology::Move m;
   m.dart = *it;
   m.dir = dirErosion;
   geo->eventQueue.push_back(m);
  }
 }
}
```

This function simply defines `dirErosion` as a vertical direction, then vertices that were previously marked in Moka as in Fig. 3 (top) are given this direction and added to the event queue. Now we must define what happens when a moving vertex collides with the top face of the lower volume:

```
void ScenarioDigging::caseVertexFace(Geomorphology::Move m) {
 list<Geomorphology::Move> newVertices;
 CDart* d = m.dart; // moving vertex
 CDart* obj = m.obj; // colliding face
 CVertex dir = m.dir;
 // topological operation
 geo->vertexFaceCollision(d,obj,&newVertices);
 CVertex initial = *map->findVertex(d);
 // process new vertices
 while(!newVertices.empty()) {
  Geomorphology::Move m2 = newVertices.front();
  // define a radial direction around the moving vertex
  m2.dir = (*map->findVertex(m2.dart) - initial).normalized() * 0.05f;
  *map->findVertex(m2.dart) += m2.dir * 0.1f;
  geo->nextStep.push_back(m2);
  newVertices.pop_front();
 }
}
```

New vertices obtained from the subroutine `vertexFaceCollision` are given a radial direction around the colliding vertex. Tweaking the hard coded value `0.1f` allows to speed up or slow down their motion at subsequent steps, which will be processed when these vertices pop out of the event queue. This parameter could also be linked to a configuration file or a slider component in the GUI to allow modification at run-time.

4.2. Cave formation

Following the previous example, a cave can be created by enlarging the hole in a user-defined lateral direction into the upper volume, as shown on Fig. 5. This is illustrated in the following algorithm where the hole will progressively extend to the right by defining this direction as the vector $(1,0,0)$.

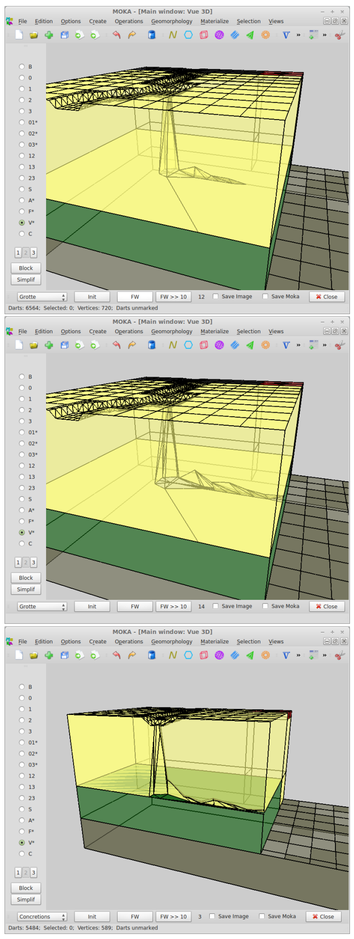

MOKA - [Main window: Vue 3D]
File Edition Options Create Operations Geomorphology Materialize Selection Views
Block
Simplif
Grotte
Concretions
Init
FW
FW >> 10
Save Image
Save Moka
Close

Figure 5: *Enlarging a hole to create a cave*

In this case we must modify the previous function to add a lateral direction and choose a new vertex to progressively create the cave. This can actually be done by slowing down the radial motion of all new vertices except one.

```
void ScenarioCave::caseVertexFace(Geomorphology::Move m) {
 ....
 // define a lateral direction
 CVertex dirCave = CVertex(1,0,0);
 // define and modify the lateral motion
 list<Geomorphology::Move>::iterator it = newVertices.begin();
 for(it; it != newVertices.end(); ++it) {
  CVertex dir2 = *map->findVertex((*it).dart) - initial;
  (*it).dir = dir2.normalized();
  // if the radial direction for this vertex is not "close" to dirCave:
  if( !(dotProduct(dirCave.normalized(), (*it).dir) > 0.8) )
   (*it).dir *= 0.4; // we simply slow down its motion
  }
}
```

4.3. Stalactites and stalagmites

In this experiment, erosion and sedimentation combine to create a single column. First of all, two vertices moving vertically are selected at the top and at the bottom of the cave (which is not necessarily closed as depicted in Fig. 6). As the animation runs, these vertices may actually collide. This is handled by the following code:

```
void ScenarioColumn::caseVertexVertex(Geomorphology::Move m) {
 list<Geomorphology::Move> newVertices;
 CDart* d = m.dart; // moving vertex
 CDart* obj = m.obj; // colliding vertex
 // make sure we do not process the colliding vertex (who could also appear in the event queue)
 int mark = map->getNewMark();
 map->markVertex(obj,mark);
 geo->switchIfMarked(mark);
 map->unmarkVertex(obj,mark);
 map->freeMark(mark);
 geo->vertexVertexCollision(d,obj,&newVertices);
 // create new vertices with radial motions
   ....
}
```

The last part of this routine, which is designed to enlarge the column, is not shown here for the sake of clarity but it is very similar to the case of a hole colliding with a face.

4.4. A complete example

By combining these different functions and by selecting moving vertices carefully, our experimental user was able to create a complete example starting with a single hole, then a cave enlarging as shown on Fig. 5, and finally small stalactites and stalagmites appearing in the cave as depicted on Fig. 7 (top). This setup was inspired by a typical illustration that is used to explain differential erosion to geology students shown on Fig. 1 (bottom).

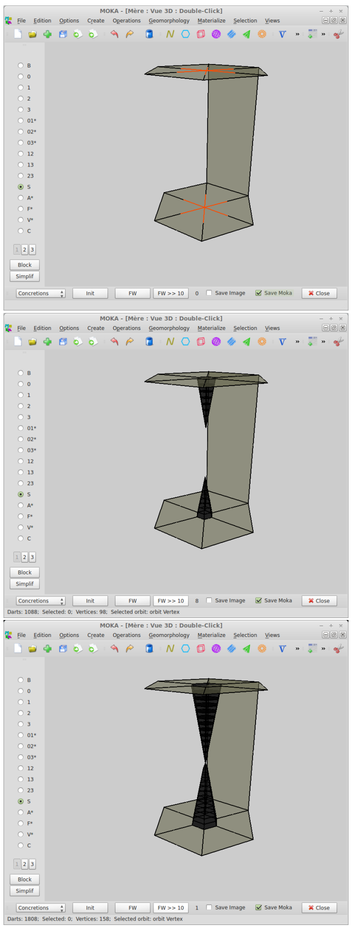
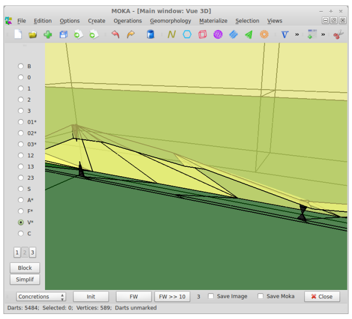


Figure 6: *Fusion of a stalactite with a stalagmite*

5. Conclusion and future work

Our framework is designed as a versatile tool to implement various scenarios and produce visual illustrations for a geomorphology class. The user can interact with the model in Moka, and implement and compile evolution functions in a standard C++ integrated development environment. The preliminary results presented in this paper are encouraging: our experimenter, who was not familiar with topological modeling, was really interested in understanding the various problems that can arise from the displacement of vertices inside a 3D model. Although for now it is almost impossible for a non-computer scientist to implement a given scenario without our help, we are currently adding new simulations that could be used during the next semesters in the class, and also to enroll other geology teachers in our experiments. It will then be possible to conduct a full-scale evaluation of the impact of our 3D simulations on geology students, based on grades and guided interviews.

Several directions for future work can be considered. The



Darts: 5484; Selected: 0; Vertices: 589; Darts unmarked

Figure 7: *Stalactites and stalagmites inside the cave shown on Fig. 5.*

first one would be to develop a simplified API that will hide most of the complex operations, as described in [SCN12] for GPU programming. This solution would certainly increase the autonomy of the non specialists and give more visibility to our work, but it is still a challenge to find the correct balance between a complex library and a simple API with limited possibilities.

Another promising direction consists in integrating other types of interactions. One of them is *hydraulic erosion*: since most of the examples presented in this paper are actually caused by the action of water, it could be interesting to add a fluid simulation to automatically select moving points and drive the animation in a more realistic way. This would also offer new directions for geomorphology research (for example to study the accumulation of sediments and the environmental impact of a retaining dam). We could also try to implement different types of faults or fractures to create earthquakes and subduction rocks as in [LHV12].

Finally, different desirable features could be added to obtain a visual result closer to the usual representations shown in Fig. 1, such as the export of the 3D model at each step towards another rendering pipeline or the possibility to include annotations.

Acknowledgements

The authors wish to thank Dr Alexandra Courtin-Nomade (University of Limoges) for her participation to this project.

References

[FP75] FREEDMAN D. P., PLUM T.: Computer programming fundamentals for non-computer scientists. In *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition* (1975), AFIPS '75, pp. 665–667. 2

[HM06] HASSINEN M., MÄYRÄ H.: Learning programming by programming: A case study. In *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006* (2006), Baltic Sea '06, pp. 117–119. 2

[LHV12] LIDAL E. M., HAUSER H., VIOLA I.: Geological storytelling: Graphically exploring and communicating geological sketches. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling* (2012), SBIM '12, Eurographics Association, pp. 11–20. 2, 7

[Lie94] LIENHARDT P.: N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *Int. J. Comput. Geometry Appl. 4*, 3 (1994), 275–324. 3

[Lie13] LIEW C. W.: Benefits of having students develop software for other departments. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education* (2013), ITiCSE '13, pp. 348–348. 2

[MB05] MCGRATH M. B., BROWN J. R.: Visual learning for science and engineering. *IEEE Computer Graphics and Applications 25*, 5 (2005), 56–63. 2

[PGMG09] PEYTAVIE A., GALIN E., MERILLOU S., GROSJEAN J.: Arches: a Framework for Modeling Complex Terrains. *Computer Graphics Forum (Proceedings of Eurographics) 28*, 2 (2009), 457–467. 2

[PJR*11] PRABHU P., JABLIN T. B., RAMAN A., ZHANG Y., HUANG J., KIM H., JOHNSON N. P., LIU F., GHOSH S., BEARD S., OH T., ZOUFALY M., WALKER D., AUGUST D. I.: A survey of the practice of computational science. In *State of the Practice Reports* (2011), SC '11, pp. 19:1–19:12. 2

[RF06] REAS C., FRY B.: Processing: programming for the media arts. *AI Soc. 20*, 4 (Aug. 2006), 526–538. 2

[SCN12] STROMME A., CARLSON R., NEWHALL T.: Chestnut: A gpu programming language for non-experts. In *Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores* (2012), PMAM '12, ACM, pp. 156–167. 2, 7

[SVBCS13] SULTANUM N., VITAL BRAZIL E., COSTA SOUSA M.: Navigating and annotating 3d geological outcrops through multi-touch interaction. In *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces* (2013), ITS '13, pp. 345–348. 2

[VD] VIDIL F., DAMIAND G.: Moka - a topology-based 3d geometric modeler. URL: `http://moka-modeller.sourceforge.net/`. 3