ReproZip: Packing Experiments for Sharing and Publication

Fernando Chirigati — NYU-Poly — fchirigati@nyu.edu
Dennis Shasha — NYU — shasha@cs.nyu.edu
Juliana Freire — NYU-Poly — juliana.freire@nyu.edu

The Need for Computational Reproducibility

- The standard of having reproducible experiments, a long tradition in natural science, has not been applied for computational science
- Researchers often have to rely on figures, plots and tables presented in papers, which loosely describe the results – these results are then difficult to reproduce [1], leading us to a credibility crisis in computational science [2]
- There are two main reasons why reproducibility is often not adopted:
  - Authors find it difficult to generate a compendium for their experiment
  - Reviewers and collaborators have difficulties trying to reproduce and verify the results, even when the compendium is available

What is ReproZip ?

ReproZip is a *general* tool for packing reproducible research. ReproZip

- tracks operating system calls and *creates a reproducible package from command-line executions*, in the author's environment *E*, with all the required files to run the experiment (***packing*** **step**). Authors do not need to port their experiment to a specific tool;
- generates a workflow specification to help reviewers and collaborators explore and verify the results, facilitating the review process;
- extracts files and workflow on another environment *E'*, without interfering with this environment (***unpacking*** **step**).

Provenance of the computational process is stored as a workflow, which can be used to: (1) reproduce results, (2) vary input parameters, (3) document the verification process and (4) support the communication between authors and reviewers/ collaborators

ReproZip does not add runtime overheads when reproducing the results, and the system has been developed for Linux distributions – successfully tested on Ubuntu and Fedora

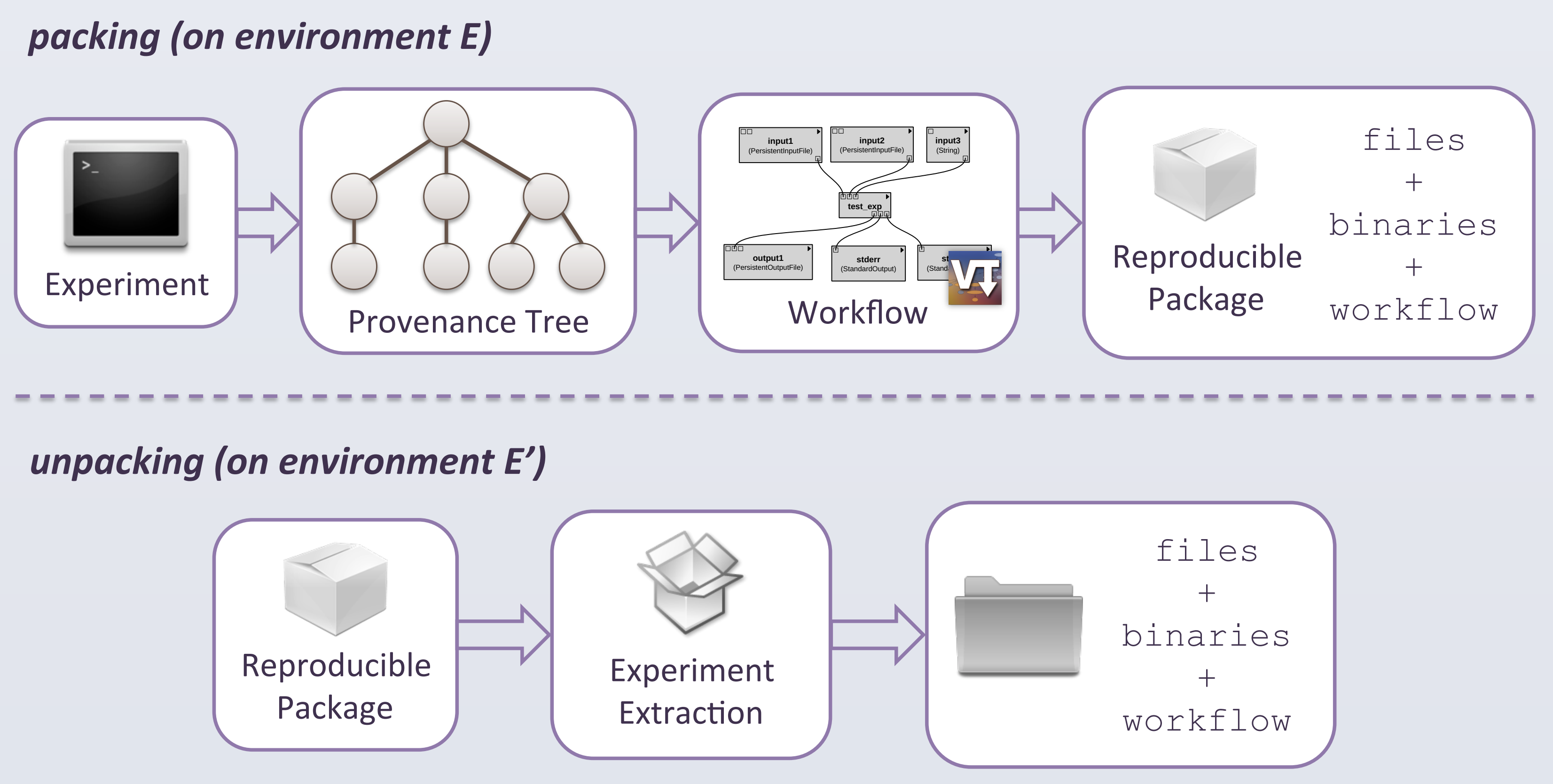
packing (on environment E)
Experiment
Provenance Tree
Workflow
Reproducible Package
files + binaries + workflow
unpacking (on environment E')
Reproducible Package
Experiment Extraction
files + binaries + workflow

Figure **1**. Packing and unpacking experiments with ReproZip

How does ReproZip Work ?

Packing Step

- ReproZip transparently captures the provenance of the execution of the experiment
  - It uses SystemTap [3] to trace system calls and capture all the required information to correctly reproduce the experiment
- The execution trace is stored in MongoDB [4], a NoSQL database
- ReproZip uses the trace data to create a *provenance tree*
- Each node in the tree stores information about an OS process, such as *files read*, *files written* and *command-line arguments* – if a process *p* spawns a process *p'*, an edge is inserted between their corresponding nodes
- The provenance tree is traversed to identify *programs*, *input files*, *output files* and *dependencies*
- The identified binaries and files are used to derive a workflow specification for the experiment
- All the required files, together with the workflow, are packed on the author's environment *E* using the *original* directory structure

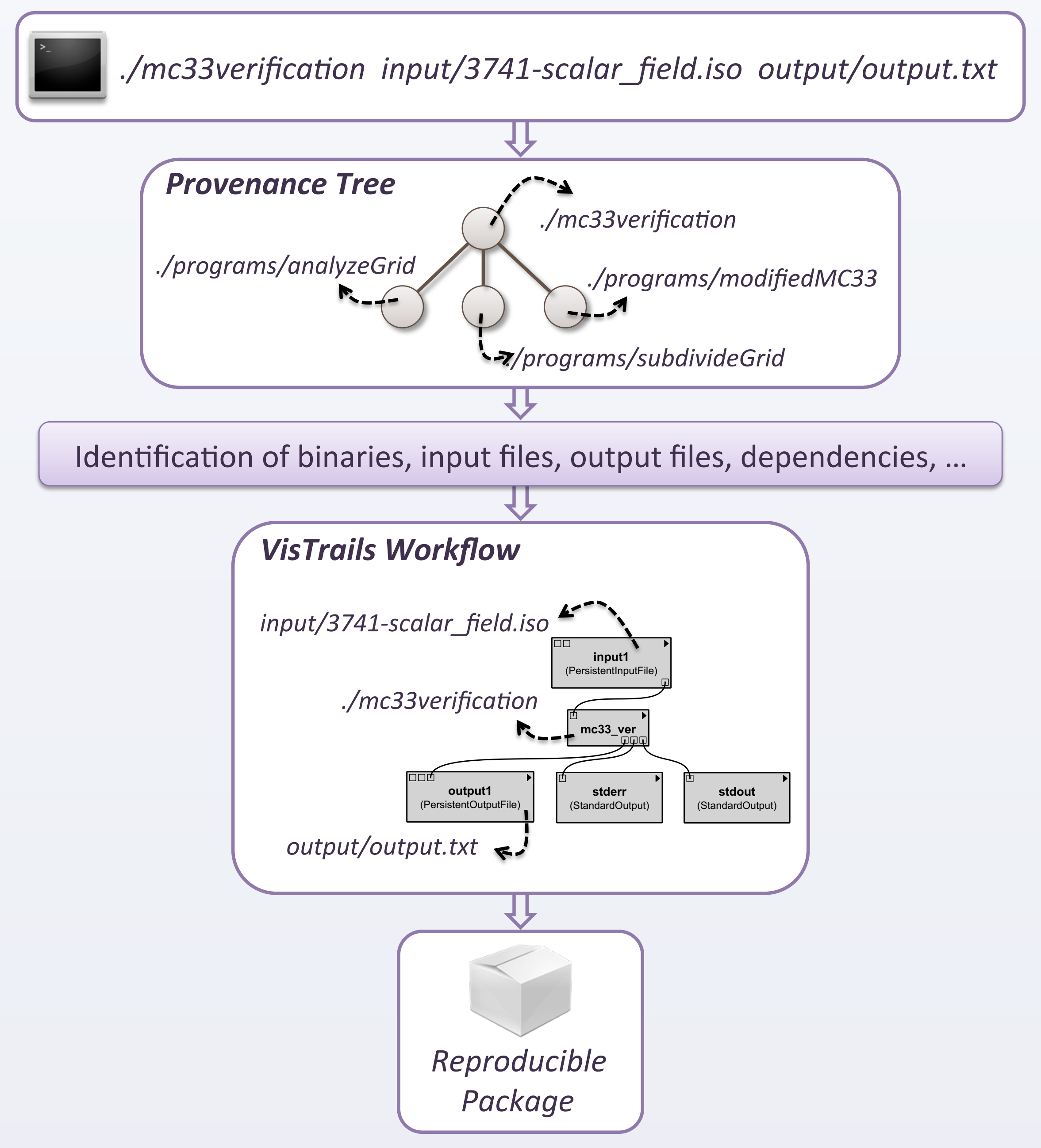

Figure **2**. Packing an experiment on topological correctness of marching cubes

Unpacking Step

- Given a reproducible package generated on environment *E*, ReproZip extracts all the files in a single directory on environment *E'*
- The workflow is configured to point to the correct files inside the unpacked directory

Verification and Exploration

- ReproZip derives a workflow specification that can be run on the VisTrails system [5]
- There are several benefits by using the VisTrails workflow: (1) experiment execution is straightforward; (2) a visual representation helps the reviewer to understand and explore the experiment; (3) users can try different parameters, as well as perform parameter sweeps and compare the results side by side; (4) users can extend the original experiment and perform additional analyses (Figure 3); and (5) the provenance of the verification process is automatically captured by VisTrails.

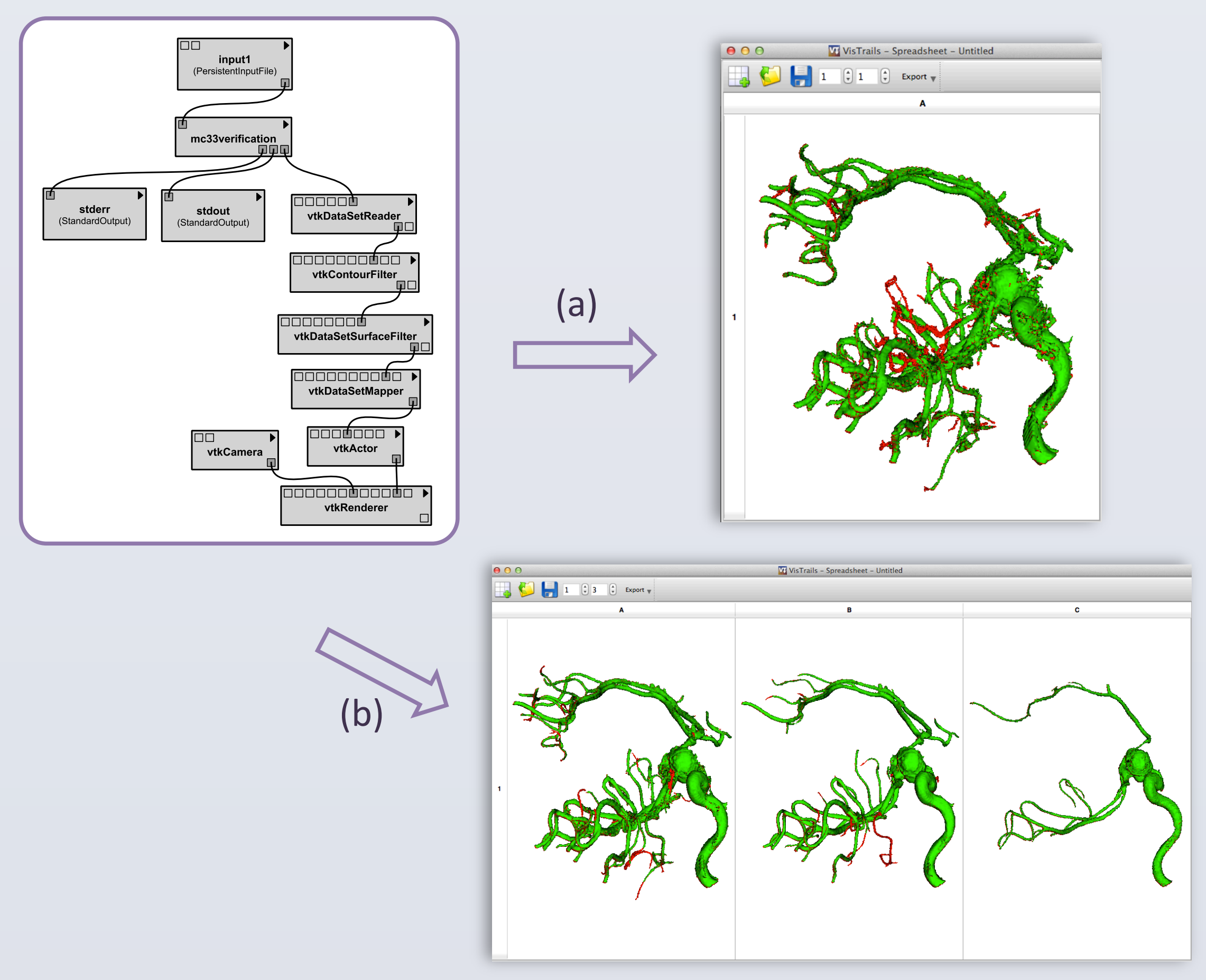

Figure **3**. Verifying the topological correctness of a marching cubes algorithm. The reviewer can (a) reproduce the results derived by the author and also (b) verify the robustness of the algorithm by exploring different isosurface values using the parameter sweep feature of VisTrails.

Acknowledgments

We thank Jesse Lingeman for feedback on our initial ideas. We also thank Lis Custódio and Tiago Etiene for providing their marching cubes experiment. This work has been partially funded by the National Science Foundation grants IIS-1139832, IIS-1142013, and CNS-1229185.

References

[1] R. LeVeque. Python tools for reproducible research on hyperbolic problems. Computing in Science & Engineering, 11(1):19{27, Jan.-Feb. 2009.

[2] D. Donoho, A. Maleki, I. Rahman, M. Shahram, and V. Stodden. Reproducible research in computational harmonic analysis. Computing in Science & Engineering, 11(1):8{18, Jan.-Feb. 2009.

[3] SystemTap. *http://sourceware.org/systemtap/*

[4] MongoDB. *http://www.mongodb.org/*

[5] J. Freire, D. Koop, E. Santos, C. Scheidegger, C. Silva, and H. T. Vo. The Architecture of Open Source Applications, chapter VisTrails. Lulu.com, 2011.