ISSN 1213-6972 *Volume 25, Number 1, 2017*

Journal of WSCG

An international journal of algorithms, data structures and techniques for computer graphics and visualization, surface meshing and modeling, global illumination, computer vision, image processing and pattern recognition, computational geometry, visual human interaction and virtual reality, animation, multimedia systems and applications in parallel, distributed and mobile environment.

Vaclav Skala – Union Agency

Journal of WSCG

Editor-in-Chief: Vaclav Skala
c/o University of West Bohemia
Faculty of Applied Sciences
Univerzitni 8
CZ 306 14 Plzen
Czech Republic
http://www.VaclavSkala.eu

Managing Editor: Vaclav Skala

Printed and Published by:
Vaclav Skala - Union Agency
Na Mazinach 9
CZ 322 00 Plzen
Czech Republic

Hardcopy: ***ISSN 1213 – 6972***
CD ROM: ***ISSN 1213 – 6980***
On-line: ***ISSN 1213 – 6964***

Journal of WSCG

Editor-in-Chief

Vaclav Skala

c/o University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering
Univerzitni 8
CZ 306 14 Plzen
Czech Republic

http://www.VaclavSkala.eu
Journal of WSCG URLs: http://www.wscg.eu or http://wscg.zcu.cz/jwscg

Editorial Advisory Board
MEMBERS

WSCG 2017

Board of Reviewers

Aburumman, Nadine (France)
Adzhiev, Valery (United Kingdom)
Anderson, Maciel (Brazil)
Assarsson, Ulf (Sweden)
Averkiou, Melinos (Cyprus)
Ayala, Dolors (Spain)
Benes, Bedrich (United States)
Bilbao, Javier,J. (Spain)
Capobianco, Antonio (France)
Carmo, Maria Beatriz (Portugal)
Charalambous, Panayiotis (Cyprus)
Cline, David (United States)
Daniel, Marc (France)
Daniels, Karen (United States)
de Geus, Klaus (Brazil)
De Martino, Jose Mario (Brazil)
de Souza Paiva, Jose Gustavo (Brazil)
Diehl, Alexandra (Germany)
Dokken, Tor (Norway)
Dong, Yue (China)
Drechsler, Klaus (Germany)
Eisemann, Martin (Germany)
Feito, Francisco (Spain)
Ferguson, Stuart (United Kingdom)
Galo, Mauricio (Brazil)
Garcia-Alonso, Alejandro (Spain)
Gdawiec, Krzysztof (Poland)
Giannini, Franca (Italy)
Gobbetti, Enrico (Italy)
Gobron, Stephane (Switzerland)
Goncalves, Alexandrino (Portugal)
Gonzalez, Pascual (Spain)
Gudukbay, Ugur (Turkey)
Hernandez, Benjamin (United States)
Jones, Mark (United Kingdom)
Juan, M.-Carmen (Spain)
Kenny, Erleben (Denmark)
Kim, HyungSeok (Korea)
Kim, Jinman (Australia)
Kurillo, Gregorij (United States)
Kurt, Murat (Turkey)
Lee, Jong Kwan Jake (United States)
Liu, Yang (China)
Liu, Beibei (United States)
Liu, SG (China)
Liu, Damon Shing-Min (Taiwan)
Lobachev, Oleg (Germany)
Luo, Shengzhou (Ireland)
Marques, Ricardo (Spain)
Mei, Gang (China)
Mellado, Nicolas (France)
Meng, Weiliang (China)
Mestre, Daniel,R. (France)
Meyer, Alexandre (France)
Molina Masso, Jose Pascual (Spain)
Molla, Ramon (Spain)
Montrucchio, Bartolomeo (Italy)
Muller, Heinrich (Germany)
Murtagh, Fionn (United Kingdom)
Nishio, Koji (Japan)
Oberweger, Markus (Austria)
Oyarzun Laura, Cristina (Germany)
Pan, Rongjiang (China)
Pedrini, Helio (Brazil)
Pereira, Joao Madeiras (Portugal)
Pina, Jose Luis (Spain)
Platis, Nikos (Greece)
Puig, Anna (Spain)
Raidou, Renata Georgia (Austria)
Ramires Fernandes, Antonio (Portugal)

Richardson, John (United States)
Ritter, Marcel (Austria)
Rodrigues, Joao (Portugal)
Rojas-Sola, Jose Ignacio (Spain)
Sanna, Andrea (Italy)
Schwaerzler, Michael (Austria)
Segura, Rafael (Spain)
Serano, Ana (Spain)
Sik-Lanyi, Cecilia (Hungary)
Sommer, Bjorn (Germany)
Sousa, A.Augusto (Portugal)
Szecsi, Laszlo (Hungary)
Teschner, Matthias (Germany)
Todt, Eduardo (Brazil)
Tokuta, Alade (United States)
Tytkowski, Krzysztof (Poland)
Umetani, Nobuyuki ()
Umlauf, Georg (Germany)
Vanderhaeghe, David (France)
Vidal, Vincent (France)
Vierjahn, Tom (Germany)
Wu, Shin-Ting (Brazil)
Wuensche, Burkhard,C. (New Zealand)
Wuethrich, Charles (Germany)
Yao, Junfeng (China)
Yoshizawa, Shin (Japan)
YU, Qizhi (United Kingdom)
Zhao, Qiang (China)

Journal of WSCG

Vol.25, No.1, 2017

Contents

Mastmeyer,A., Wilms,M., Handels,H.: Interpatient Respiratory Motion Model Transfer for Virtual Reality Simulations of Liver Punctures 1

Nysjo,F., Olsson,P., Malmberg,F., Carlbom,I.B., Nystrom,I.: Using Anti-Aliased Signed Distance Fields for Generating Surgical Guides and Plates from CT Images 11

Vintescu,A.-M., Dupont,F., Lavoue,G., Pooran,M., Tierny,J.: Least Squares Affine Transitions for Global Parameterization 21

Gerrits,T., Roessl,C., Theisel,H.: Glyphs for Space-Time Jacobians of Time-Dependent Vector Fields 31

Ben Salah,F., Belhaouari, H., Arnould,A., Meseure,P.: A Modular Approach Based On Graph Transformation To Simulate Tearing And Fractures On Various Mechanical Models 39

Lousada,P., Costa,V., Pereira,J.M.: Bandwidth and Memory Efficiency in Real-Time Ray Tracing 49

Soares,A.S., Apolinario,A L.: Real-time 3D Gesture Recognition using Dynamic Time Warping and Simplification Methods 59

Stamatakis,D., Benger,W., Shrira,L.: Flexible navigation through a multi-dimensional parameter space using Berkeley DB snapshots 67

Interpatient Respiratory Motion Model Transfer for Virtual Reality Simulations of Liver Punctures

Andre Mastmeyer
Univ. of Luebeck
Inst. of Med. Inform.
Ratzeburger Allee 160
23568 Luebeck,
Germany
mastmeyer@imi.uni-luebeck.de

Matthias Wilms
Univ. of Luebeck
Inst. of Med. Inform.
Ratzeburger Allee 160
23568 Luebeck,
Germany
wilms@imi.uni-luebeck.de

Heinz Handels
Univ. of Luebeck
Inst. of Med. Inform.
Ratzeburger Allee 160
23568 Luebeck,
Germany
handels@imi.uni-luebeck.de

ABSTRACT

Current virtual reality (VR) training simulators of liver punctures often rely on static 3D patient data and use an unrealistic (sinusoidal) periodic animation of the respiratory movement. Existing methods for the animation of breathing motion support simple mathematical or patient-specific, estimated breathing models. However with personalized breathing models for each new patient, a heavily dose relevant or expensive 4D data acquisition is mandatory for keyframe-based motion modeling. Given the reference 4D data, first a model building stage using linear regression motion field modeling takes place. Then the methodology shown here allows the transfer of existing reference respiratory motion models of a 4D reference patient to a new static 3D patient. This goal is achieved by using non-linear inter-patient registration to warp one personalized 4D motion field model to new 3D patient data. This cost- and dose-saving new method is shown here visually in a qualitative proof-of-concept study.

Keywords

Virtual Reality, Liver Puncture Training, 4D Motion Models, Inter-patient Registration of Motion Models

1 INTRODUCTION

The virtual training and planning of minimally invasive surgical interventions with virtual reality simulators provides an intuitive, visuo-haptic user interface for the risk-sensitive learning and planning of interventions. The simulation of liver punctures has been an active research area for years [For16, For15, Mas14].

Obviously first, the stereoscopic visualization of the anatomy of the virtual patient body is important [For12]. Second, the haptic simulation of the opposing forces through the manual interaction, rendered by haptic input and output devices, with the patient is key [For13]. Third in recent developments, the simulation of the appearance and forces of the patient's breathing motions is vital [Mas17, Mas14].

The previously known VR training simulators usually use time invariant 3D patient models. A puncture of the spinal canal can be simulated sufficiently plausibly by such models. In the thoracic and upper abdominal region, however, respiratory and cardiac movements are constantly present. In the diaphragm area at the bottom of the lungs just above the liver, breathing movement differences in the longitudinal z direction of up to 5 cm were measured [Sep02]. Now for 4D animation, the necessary data consists of a single 3D CT data set and a mathematical or personalized animation model. Our aim here is to incorporate these physiological-functional movements into realistic modeling in order to

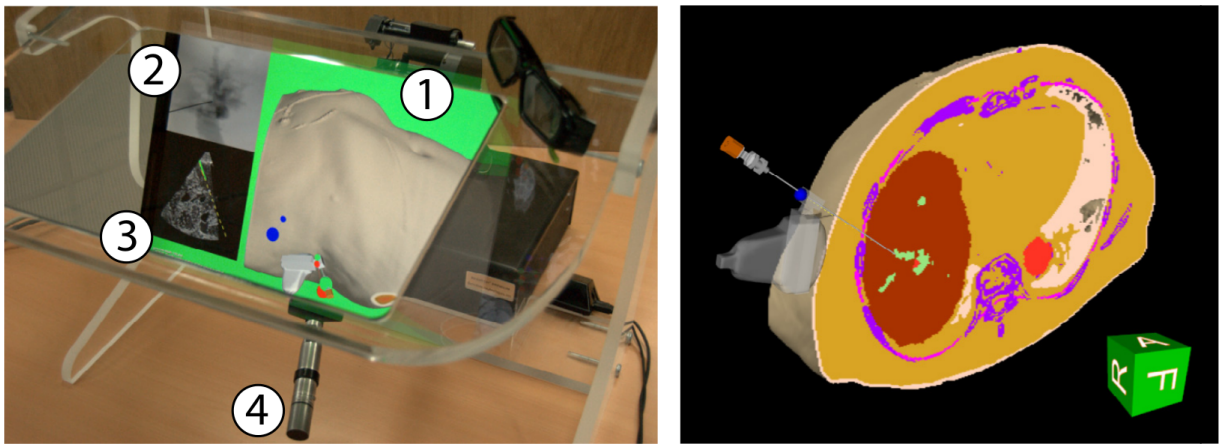


Figure 1: Left: Hardware: (1) Main stereo rendering window with successful needle insertion into a target, (2) fluoroscopic simulation, (3) Ultrasound simulation, (4) haptic device handle. Right: Main rendering window displaying oblique cut and color-coded patient structures just before needle insertion into a targeted bile duct (green).

offer the user a more realistic visuo-haptic VR puncture simulation. This means also to take into account the intra- and intercycle variability (hysteresis, variable amplitude during inhalation / exhalation).

A major interest and long term goal of virtual and augmented reality is the planning [Rei06] and intra-operative navigation assistance [Nic05]. However, in these works breathing motion is not incorporated or applicability limits by neglecting breathing motion in terms of minimal tumor size are given [Nic05]. Published approaches from other groups [Vil14, Vil11] model only a sinusoidal respiratory motion without hysteresis and amplitude variation. First steps in the direction of a motion model building framework were taken by our group [Ehr11]. Accurate simulation of respiratory motion depending on surrogate signals is relevant e.g. in fractionated radiotherapy. However, since a patient-specific 4D volume data set is required for personalized breathing model building and its acquisition is associated with a high radiation dose with 4D-CT ($\geq$ 20-30 mSv (eff.)), our approach is the transfer of existing 4D breathing models to new 3D patient data. For comparison, the average natural background radiation is approximately 2.1 mSv (eff.) per year[1].

On the other hand, there is no medical indication to acquire 4D CT data just for training purposes and model building from 4D MR data to be included is unjustifiable for cost reasons.

In this paper, we present a feasibility study with first qualitative results for the transfer of an existing 4D breathing model [Wil14] to static 3D patient data, in which only a 3D CT covering chest and upper abdomen at maximum inhalation is necessary (approximately 2- 13 mSv (eff.))[2].

2 RECENT SOLUTION

The existing solution requires a full 4D data set acquisition for each new patient. In [For16, Mas16, Mas13, For14], concepts for a 3D VR simulator and efficient patient modeling for the training of different punctures (e.g. liver punctures) have already been presented, see Fig. 1. A Geomagic Phantom Premium 1.5 High-Force is used for the manual control and haptic force feedback of virtual surgical instruments. Nvidia shutter glasses and a stereoscopic display provide the plausible rendering of the simulation scene. This system uses time invariant 3D CT data sets as a basis for the patient model. In case of manual interaction with the model, tissue deformation due to acting forces of the instruments are represented by a direct visuo-haptic volume rendering method.

New developments of VR simulators [For15] allow a time-variant 4D-CT data set to be used in real time for the visualized patient instead of a

[1] Intercontinental flight max. 0.11 mSv (eff.)

[2] Siemens Somatom Definition AS

static 3D CT data set. The respiratory movement can be visualized visuo-haptically as a keyframe model using interpolation or with a flexible linear regression based breathing model as described below.

3 PROPOSED SOLUTION

The new solution requires only a 3D data set acquisition for each new patient.

3.1 Modeling of Breathing Motion

Realistic, patient-specific modeling of breathing motion in [For15] relies on a 4D CT data set covering one breathing cycle. It consists of N_{phases} phase 3D images indexed by j. Furthermore, a surrogate signal (for example, acquired by spirometry) to parametrize patient breathing in a low-dimensional manner is necessary.

We use a measured spirometry signal $v(t)$ [ml] and its temporal derivative in a composite surrogate signal: $(v(t), v'(t))^T$. This allows to describe different depths of breathing and the distinction between inhalation and exhalation (respiratory hysteresis). We assume linearity between signal and motion extracted from the 4D data. First, we use the 'sliding motion'-preserving approach from [SR12] for $N_{phases} - 1$ intra-patient inter-phase image registrations to a selected reference phase j_{ref}:

$$\varphi_j^{pat4D} = \underset{\varphi}{\operatorname{argmin}} \left(D_{NSSD} \left[I_j^{pat4D}, I_{j_{ref}}^{pat4D} \circ \varphi \right] + \alpha_S \cdot R_S(\varphi) \right), \quad (1)$$

$$j \in \{1, .., j_{ref} - 1, j_{ref} + 1, .., N_{phases}\},$$

where a distance measure D_{NSSD} (normalized sum of voxel-wise squared differences [Thi98]) and a specialized regularization R_S establishes smooth voxel correspondences except in the pleural cavity where discontinuity is a wished feature [SR12]. Based on the results, the coefficients $a_{1..3}^{pat4D}$ are estimated as vector fields over the positions $\mathbf{x}$. The personalized breathing model then can be stated as a linear multivariate regression [Wil14]:

$$\begin{aligned} \hat{\varphi}^{pat4D}(\mathbf{x}, t) = & a_1^{pat4D}(\mathbf{x}) \cdot v(t) + \\ & a_2^{pat4D}(\mathbf{x}) \cdot v'(t) + \\ & a_3^{pat4D}(\mathbf{x}),\ \mathbf{x} \in \Omega_{pat4D}. \end{aligned} \quad (2)$$

Thus, a patient's breathing state can be represented by a previously unseen breathing signal: Any point in time t corresponds to a shifted reference image $I_{j_{\text{ref}}}^{pat4D} \circ \hat{\varphi}^{pat4D}(\mathbf{x}, t)$. Equipped with a real-time capable rendering technique via ray-casting with bent rays (see [For15] for full technical details), the now time variant model-based animatable CT data $I_{j_{\text{ref}}}^{pat4D}$ can be displayed in a new variant of the simulator and used for training. The rays are bent by the breathing motion model and this conveys the impression of an animated patient body, while being very time efficient (by space-leaping and early ray-termination) compared to deforming the full 3D data set for each time point and linear ray-casting afterwards [For15].

3.2 Transfer of Existing Respiratory Models to new, static Patient Data

Using the method described so far, personalized breathing models can be created, whose flexibility is sufficient to approximate the patients' breathing states, which are not seen in the observation phase of the model formation.

However, the dose-relevant or expensive acquisition of at least one 4D data set has thus far been necessary for each patient.

Therefore, here we pursue the idea to transfer a readily built 4D reference patient breathing model to new static patient data $pat3D$ and to animate it in the VR simulator described in Sec. 2.

For this purpose, it is necessary to correct for the anatomical differences between the reference patient with the image data $I_{j_{\text{ref}}}^{pat4D}$ and the new patient image data I_{ref}^{pat3D} based on a similar breathing phase. This is achieved, for example, by a hold-breath scan (ref) in the maximum inhalation state, which corresponds to a certain phase j_{ref} in a standardized 4D acquisition protocol. A nonlinear inter-patient registration $\varphi(\mathbf{x}) : \Omega_{pat3D} \rightarrow \Omega_{pat4D}$ with minimization of a relevant image distance D ensures the necessary compensation [Mas16, Mas13]:

$$\varphi_{j_{\text{ref}}}^{pat3D \rightarrow pat4D} = \underset{\varphi}{\operatorname{argmin}} \left(D_{SSD} \left[I_{\text{ref}}^{pat3D}, I_{j_{\text{ref}}}^{pat4D} \circ \varphi \right] + \alpha_D \cdot R_D(\varphi) \right), \quad (3)$$

where a distance measure D_{SSD} (sum of squared voxel-wise differences) and a diffusive non-linear regularization R_D establishes smooth inter-patient voxel correspondences. On both sides, the breathing phase 3D image of maximum inhalation is selected as the reference phase (ref). The distance measurement can be selected according to the modality and quality of the image data. The transformation $\varphi_{j_{\text{ref}}}^{pat3D\to pat4D}$, which is determined in the nonlinear inter-patient registration, can now be used to warp the intra-patient inter-phase deformations of the reference patient φ_j^{pat4D} as a plausible estimate φ_j^{pat3D} ($j \in \{1,\ldots,n\}$; $\circ$: right to left):

$$\varphi_j^{pat3D} = \left(\varphi_{j_{\text{ref}}}^{pat3D\to pat4D}\right)^{-1} \circ \varphi_j^{pat4D} \circ \varphi_{j_{\text{ref}}}^{pat3D\to pat4D}. \quad (4)$$

The approach for estimating the respiratory motion for the new patient can now be applied analogously to the reference patient (see Sec. 3.1). With a efficient regression method [Wil14], the breathing movement of virtual patient models, which are only based on a comparatively low dose of acquired 3D-CT data, can be plausibly approximated:

$$\begin{aligned}\hat{\varphi}^{pat3D}(\mathbf{x},t) = & a_1^{pat3D}(\mathbf{x}) \cdot v(t) + \\ & a_2^{pat3D}(\mathbf{x}) \cdot v'(t) + \\ & a_3^{pat3D}(\mathbf{x}),\ \mathbf{x} \in \Omega_{pat3D}. \end{aligned} \quad (5)$$

Optionally, simulated surrogate signals $v(t)$ can be used for the 4D animation of 3D CT data. Simple alternatives are to use the surrogate signal of the reference patient or also a (scaled) signal of the new patient *pat*3*D*, which can simply be recorded with a spirometric measuring device without new image acquisition.

4 EXPERIMENTS AND RESULTS

We performed a qualitative feasibility study, results are animated in the 4D VR training simulator [For15].

For the 4D reference patient, a 4D-CT data set of the thorax and upper abdomen with 14 respiratory phases ($512^2 \times 462$ voxel to 1^3 mm) and a spirometry signal $v(t)$ were used (Fig. 2). The new patient is represented only by a static 3D CT data set ($512^2 \times 318$ voxel to 1^3 mm).

All volume image data was reduced to a size of 256^3 voxel due to the limited graphics memory of the GPU used (Nvidia GTX 680 with 3 GB RAM).

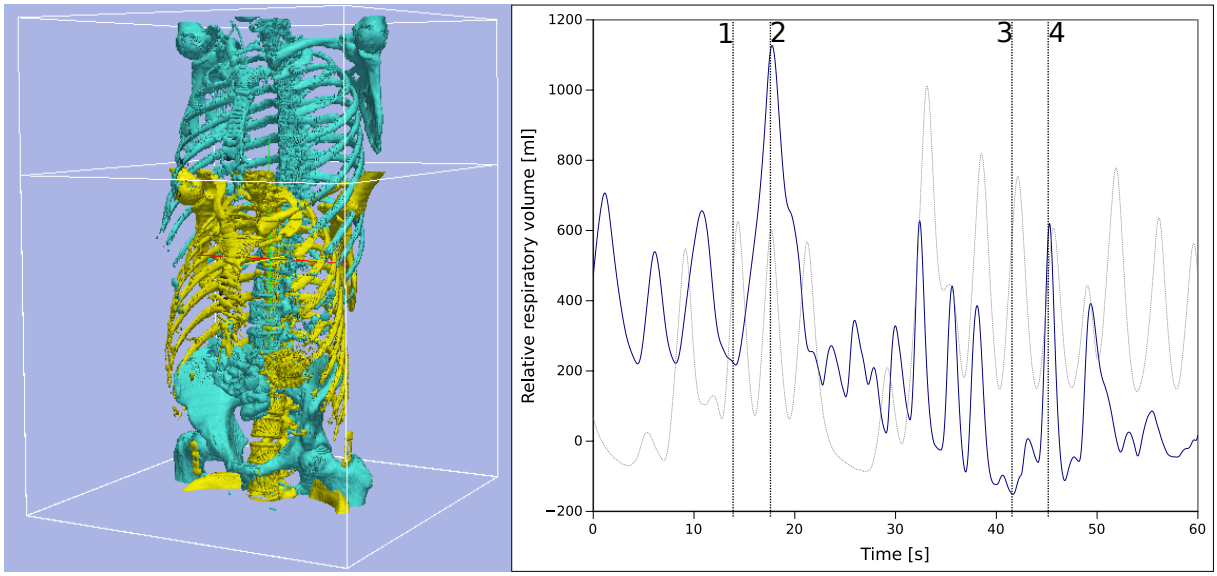
According to Eq. 1 we first perform the intra-patient inter-phase registrations to a chosen reference phase j_{ref}.

The registrations from Eqs. 1 and 3 use weights $\alpha_S = 0.1$ and $\alpha_D = 1$ for the regularizers R_S and R_D. In both registration processes, the phase with maximum inhalation is used as the reference respiratory phase j_{ref} and for the training of the breathing model.

The respiratory signal used for model training is shown in Fig. 2b, gray curve. We show the areas with plausible breathing simulation and use the unscaled respiratory signal of *pat*3*D* with larger variance to provoke artifacts (Fig. 2b, blue curve). The model training according to Eqs. 2 and 5 is very efficient using matrix computations.

We use manual expert segmentations of the liver and lungs, available for every phase of the 4D patient, to mainly assess the quality of the inter-patient registration in Eq. 3. Via the availabe inter-phase registrations φ_j^{pat4D} (Eq. 1) to the 4D reference phase, we first warp the phase segmentation masks accordingly. After applying the inter-patient registration to *pat*3*D*, we have the segmentation masks of *pat*4*D* in the space of the targeted 3D patient. Now for this patient, also a manual expert segmentation is availabe for comparison. Quantitatively, the DICE coefficients of the transferred segmentation masks (liver, lungs) can be given to classify the quality of the registration chain of the reference respiratory phases (single atlas approach). Qualitatively, we present sample images for four time instants and a movie.

The mean DICE coefficients of the single-atlas registration of the liver and lung masks to the new static patient *pat*3*D* yield satisfying values of 0.86 ± 0.12 and 0.96 ± 0.09. Note the clearly different scan ranges of the data sets (Fig. 2a). The animation of the relevant structures is shown as an example in Fig. 3, using a variable real breathing signal of the target patient *pat*3*D* (Fig. 2b). In the puncture-relevant liver region, the patient's breathing states are simulated plausibly for the 4D



(a) Field of view difference between reference patient *pat*4*D* (turquoise) and target patient *pat*3*D* (yellow).

(b) Selected times of the spirometry signal from *pat*3*D* (blue).

Figure 2: (a) Field of views and (b) respiratory signals of the patients *pat*4*D* (gray dashed) vs. *pat*3*D*.

reference patient (Fig. 3) and, more importantly, the 3D patient (Figs. 4, 5), to which the motion model of *pat*4*D* was transferred[3].

5 DISCUSSION, OUTLOOK AND CONCLUSION

For interested readers, the basic techiques for 4D breathing motion models have been introduced in [Ehr11] by our group. However there, the motion model is restricted to the inside of the lungs and by design a mean motion model is built from several 4D patients. The mean motion model is artificial to some degree, more complex and timely to build. The method described here for the transfer of retrospectively modeled respiratory motion of one 4D reference patient to a new 3D patient data set is less complex and extends to a larger body area. It already allows the plausible animation of realistic respiratory movements in a 4D-VR-training-simulator with visuo-haptic interaction. Of course in the future, we want to build a mean motion model for the whole body section including (lower) lungs and the upper abdomen, too.

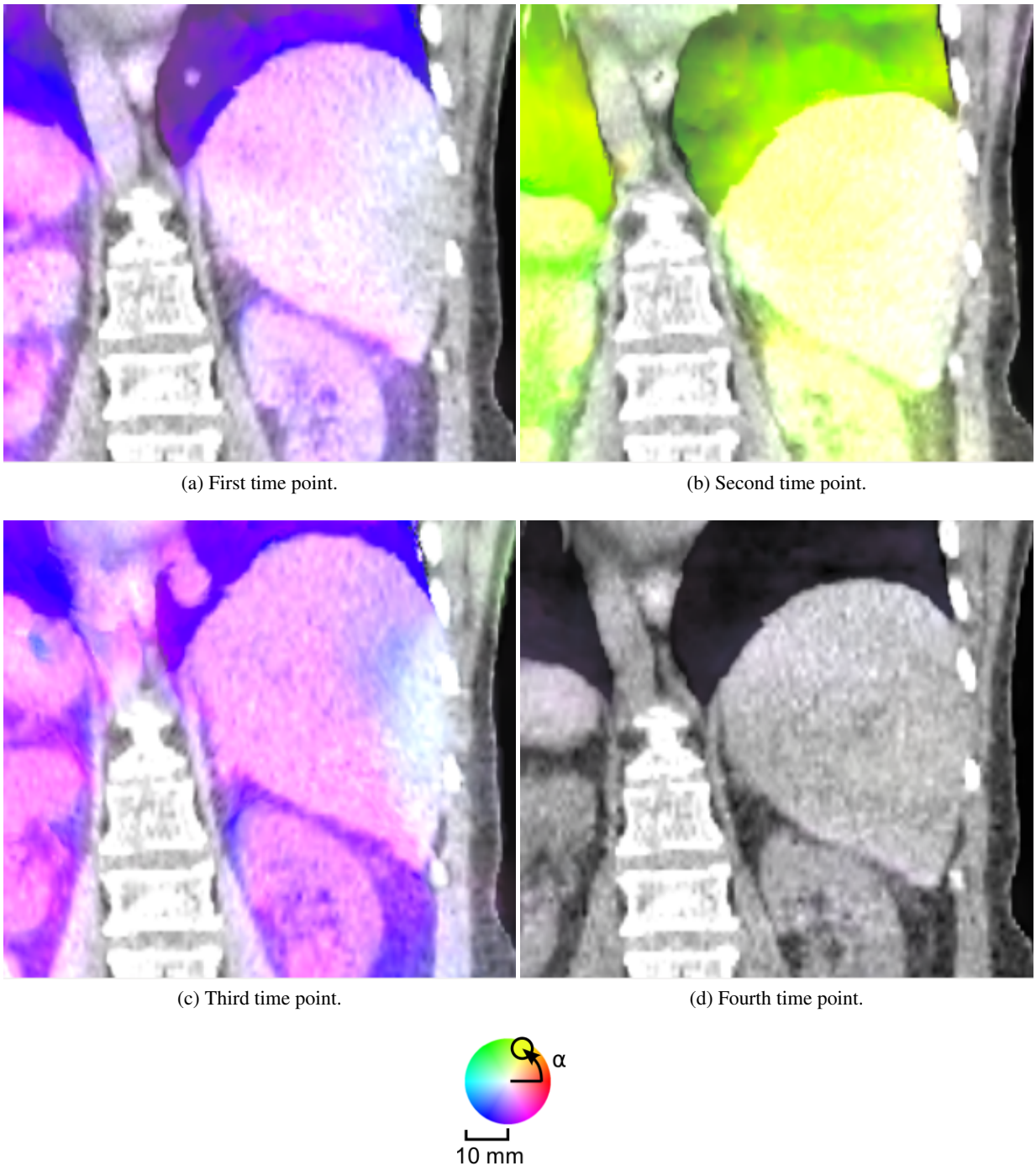
In other studies, we found $\alpha_D = 1$ in Eq. 3 robust (compromise between accuracy and smoothness) for inter-patient registration with large shape variations [Mas13, Mas16]. In Eq. 1 for intra-patient inter-phase registration, we use $\alpha_S = 0.1$ to allow more flexibility for more accuracy as the shape variation between two phases of the same patient is much smaller [SR12].

We achieve qualitatively plausible results for the liver area in this feasibility study. In the upper thorax especially at the rib cage in neighborhood to the dark lungs stronger artifacts can occur (Fig. 5c). They are due to problems in the inter-patient registration that is a necessary step for the transfer of the motion model. The non-linear deformation sometimes is prone to misaligned ribs. The same is true for the lower thorax with perforation first of the liver and then diaphragm (Fig. 4c). Further optimization have to be carried out as artifacts can appear on the high contrast lung edge (diaphragm, ribs) with a small tidal volume. For liver punctures only, the artifacts of smeared ribs are minor as can be seen in Fig. 4.

Summing up, the previous assumption from Sec. 2 of a dose-relevant or expensive acquisition of a 4D-CT data set for each patient, can be mitigated for liver punctures by the presented transfer of an existing 4D breathing model.

Future work will deal with the better adaptation and simulation of the breathing signal. Further

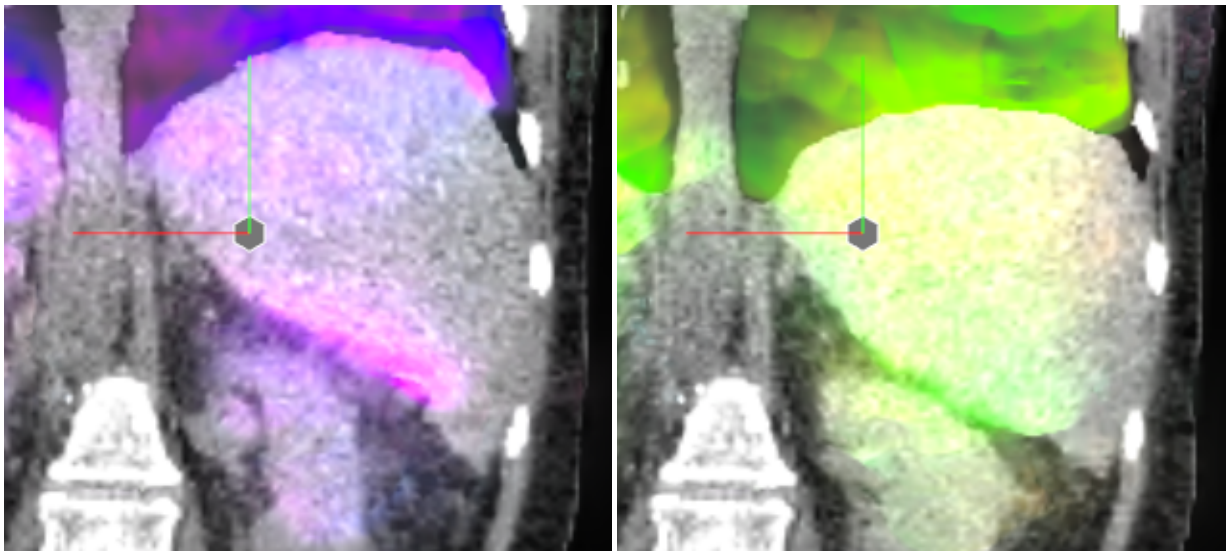
[3] Demo movie, click here



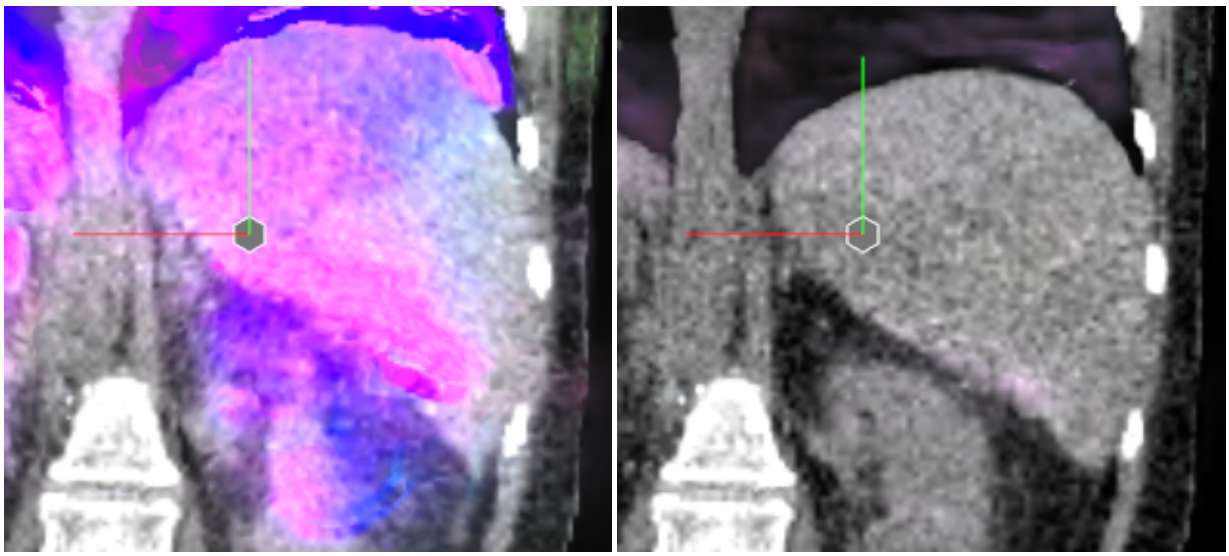
(a) First time point. (b) Second time point.

(c) Third time point. (d) Fourth time point.

Figure 3: Field of view, respiratory signal and coronal views with overlayed motion field to the CT data of the patients *pat*4*D* (a-d). The color wheel legend below indicates the direction of the motion field.



(a) First time point. (b) Second time point.



(c) Third time point. (d) Fourth time point.

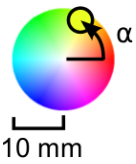


Figure 4: Coronal views with overlayed motion field to the CT data of the patient *pat*3*D* (a-d) deformed with the model of *pat*4*D*. The color wheel legend below indicates the direction of the motion field.

(a) First time point. (b) Second time point.

(c) Third time point. (d) Fourth time point.

Figure 5: Upper thorax coronal views of the animated CT data of the patient *pat*3*D* (a-d) deformed with the model of *pat*4*D*. Rib artifacts are indicated by the yellow arrow in (c).

topics are the optimization of the inter-patient registration and the construction of alternatively selectable mean 4D reference breathing models. As in [For16], the authors plan to perform usability studies with medical practitioners.

To conclude, the method allows VR needle puncture training in the hepatic area of breathing virtual patients based on a low-risk and cheap 3D data acquisition for the new patient only. The requirement of a dose-relevant or expensive acquisition of a 4D CT data set for each new patient can be mitigated by the presented concept. Future work will include the reduction of artifacts and building mean reference motion models.

6 ACKNOWLEDGEMENT

Support by grant: DFG HA 2355/11-2.

7 REFERENCES

[Ehr11] Ehrhardt, J., Werner, R., Schmidt-Richberg, A., Handels, H. Statistical modeling of 4D respiratory lung motion using diffeomorphic image registration. IEEE Transactions on Medical Imaging, 30(2):251–265, September 2011.

[For12] Fortmeier, D., Mastmeyer, A., Handels, H. GPU-based visualization of deformable volumetric soft-tissue for real-time simulation of haptic needle insertion. German Conference on Medical Image Processing BVM - 2012: Algorithms - Systems - Applications. Proceedings from 18.-20. March 2012 in Berlin, pages 117–122, 2012.

[For13] Fortmeier, D., Mastmeyer, A., Handels, H. Image-based palpation simulation with soft tissue deformations using chainmail on the GPU. German Conference on Medical Image Processing - BVM 2013, pages 140–145, 2013.

[For14] Fortmeier, D., Mastmeyer, A., Handels, H. An image-based multiproxy palpation algorithm for patient-specific VR-simulation. Medicine Meets Virtual Reality 21, MMVR 2014, pages 107–113, 2014.

[For15] Fortmeier, D., Wilms, M., Mastmeyer, A., Handels, H. Direct visuo-haptic 4D volume rendering using respiratory motion models. IEEE Trans Haptics, 8(4):371–383, 2015.

[For16] Fortmeier, D., Mastmeyer, A., Schröder, J., Handels, H. A virtual reality system for PTCD simulation using direct visuo-haptic rendering of partially segmented image data. IEEE J Biomed Health Inform, 20(1):355–366, 2016.

[Mas13] Mastmeyer, A., Fortmeier, D., Maghsoudi, E., Simon, M., Handels, H. Patch-based label fusion using local confidence-measures and weak segmentations. Proc. SPIE Medical Imaging: Image Processing, pages 86691N–1–11, 2013.

[Mas14] Mastmeyer, A., Hecht, T., Fortmeier, D., Handels, H. Ray-casting based evaluation framework for haptic force-feedback during percutaneous transhepatic catheter drainage punctures. Int J Comput Assist Radiol Surg, 9:421–431, 2014.

[Mas16] Mastmeyer, A., Fortmeier, D., Handels, H. Efficient patient modeling for visuo-haptic VR simulation using a generic patient atlas. Comput Methods Programs Biomed, 132:161–175, 2016.

[Mas17] Mastmeyer, A., Fortmeier, D., Handels, H. Evaluation of direct haptic 4d volume rendering of partially segmented data for liver puncture simulation. Nature Scientific Reports, 7(1):671, 2017.

[Nic05] Nicolau, S., Pennec, X., Soler, L., Ayache, N. A complete augmented reality guidance system for liver punctures: First clinical evaluation. Medical Image Computing and Computer-Assisted Intervention–MICCAI 2005, pages 539–547, 2005.

[Rei06] Reitinger, B., Bornik, A., Beichel, R., Schmalstieg, D. Liver surgery planning using virtual reality. IEEE Computer Graphics and Applications, 26(6):36–47, 2006.

[Sep02] Seppenwoolde, Y., Shirato, H., Kitamura, K., Shimizu, S., Herk, M.van , Lebesque, J. V., Miyasaka, K. Precise and real-time measurement of 3D tumor motion in lung due to breathing and heartbeat, measured during radiotherapy. Int J Radiation Oncololgy, Biology, Physics, 53(4):822–834, Jul 2002.

[SR12] Schmidt-Richberg, A., Werner, R., Handels, H., Ehrhardt, J. Estimation of slipping organ motion by registration with direction-dependent regularization. Medical Image Analysis, 16(1):150 – 159, 2012.

[Thi98] Thirion, J.-P. Image matching as a diffusion process: an analogy with maxwell's demons. Medical Image Analysis, 2(3):243 – 260, 1998.

[Vil11] Villard, P., Boshier, P., Bello, F., Gould, D. Virtual reality simulation of liver biopsy with a respiratory component. Liver Biopsy, InTech, pages 315–334, 2011.

[Vil14] Villard, P., Vidal, F., Cenydd, L., Holbrey, R., Pisharody, S., Johnson, S., Bulpitt, A., John, N., Bello, F., Gould, D. Interventional radiology virtual simulator for liver biopsy. Int J Comput Assist Radiol Surg, 9(2):255–267, 2014.

[Wil14] Wilms, M., Werner, R., Ehrhardt, J., et al. Multivariate regression approaches for surrogate-based diffeomorphic estimation of respiratory motion in radiation therapy. Phys Med Biol, 59:1147–1164, 2014.

Using Anti-Aliased Signed Distance Fields for Generating Surgical Guides and Plates from CT Images

Fredrik Nysjö, Pontus Olsson, Filip Malmberg, Ingrid B. Carlbom, and Ingela Nyström
Centre for Image Analysis, Dept. of Information Technology,
Uppsala University, Sweden
{fredrik.nysjo, pontus.olsson, filip.malmberg, ingrid.carlbom, ingela.nystrom}@it.uu.se

ABSTRACT

We present a method for generating shell-like objects such as surgical guides and plates from segmented computed tomography (CT) images, using signed distance fields and constructive solid geometry (CSG). We develop a user-friendly modeling tool which allows a user to quickly design such models with the help of stereo graphics, six degrees-of-freedom input, and haptic feedback, in our existing software for virtual cranio-maxiollofacial surgery planning, HASP. To improve the accuracy and precision of the modeling, we use an anti-aliased distance transform to compute signed distance field values from fuzzy coverage representations of the bone. The models can be generated within a few minutes, with only a few interaction steps, and are 3D printable. The tool has potential to be used by the surgeons themselves, as an alternative to traditional surgery planning services.

Keywords

Implicit Modeling, Distance Fields, CT, Shells, Virtual Surgery Planning

1 INTRODUCTION

Virtual surgery planning and pre-operative design and fabrication of plastic surgical guides and titanium plates for bone fixation have proven valuable for improving outcome and reducing cost in reconstructive surgery such as cranio-maxillofacial (CMF) surgery [25]. However, existing virtual surgery planning tools, such as Materialise's Mimics [14], have complex user interfaces (UIs) limited to two-dimensional (2D) interaction with three-dimensional (3D) data, such as computed tomography (CT) images, which can be non-intuitive for clinicians to use and therefore often require the help of a technician or an engineer. This includes the task of designing of patient-specific models of surgical guides and plates, something which today is often outsourced to external companies with lead times of several days or in some cases weeks.

The Haptics-Assisted Surgery Planning (HASP) system [18][19] was developed within our research group with the aim of shortening the preoperative planning time from days to hours, by providing a user-friendly software and UI that can be used by the surgeons themselves. HASP supports stereo graphics, six degrees-of-freedom (DOF) input, and haptic feedback in a software that incorporates bone, vessels, and soft tissue planning for CMF defect reconstruction. Surgeons who have tested the software have found it to be an efficient tool for planning so called fibula osteocutaneous free flap reconstructions, in which a bone graft from the fibula (calf bone) is transplanted to the defect, for example, a resected part of the mandible (lower jaw) [19].

Previous versions of the HASP software did not support the design of patient-specific plates for bone fixation, or cutting guides with slots or flanges that help the surgeon to perform precise osteotomies or resections (cuts) according to a surgical plan. In this paper, we present the tools we developed to enable surgeons to quickly design models for such parts within a few minutes, and also present a method for generating the models from segmented CT images.

1.1 Contribution

The main contributions of this paper are:

- A fast and efficient method for generating 3D printable models of surgical guides and plates from segmented CT images.
- A method for improving the accuracy and precision of the modeling by using fuzzy coverage representations in combination with an anti-aliased distance transform.

2 RELATED WORK

There are a few examples in the literature of systems or methods for helping surgeons designing surgical guides

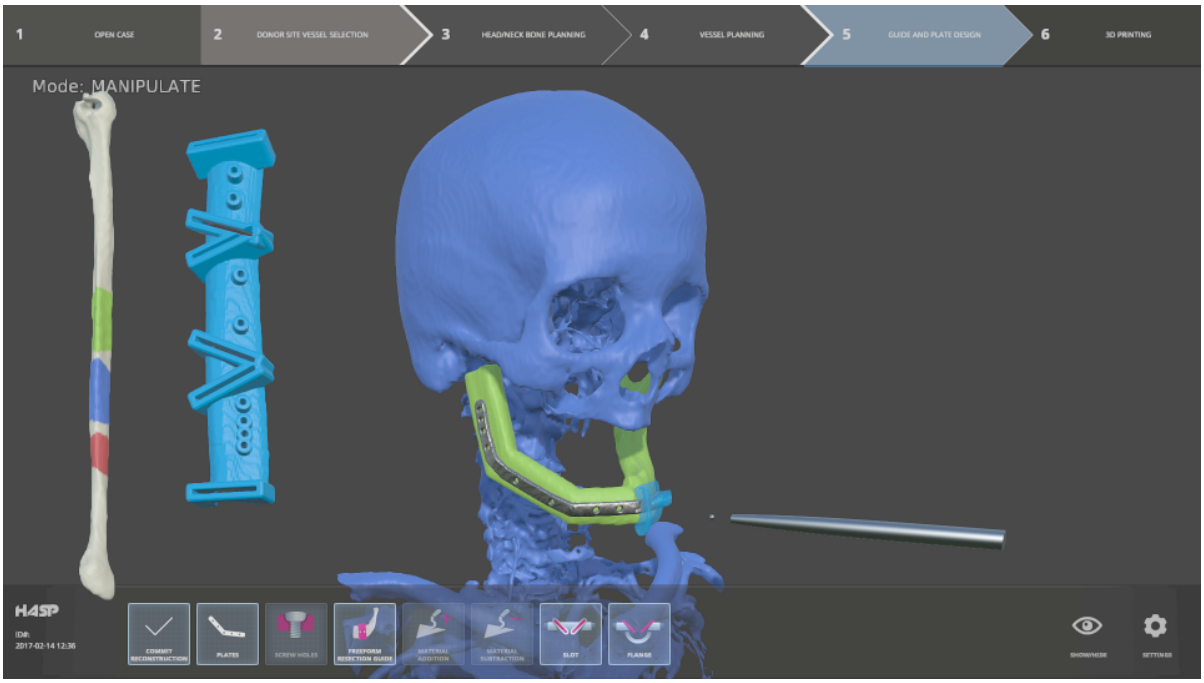


Figure 1: The HASP user interface. A six DOF haptic device controls the virtual stylus (right) used for interaction. The screenshot shows a mandible (in green) being reconstructed from the original mandible combined with a transplant from the fibula (left), and surgical guides (in blue) and plates (in metal) generated in the software.

and plates. Kovler et al. [12] describe a system combining stereo graphics and haptic feedback with a sketch-based method for plate design. However, their system is aimed at trauma surgery and does not provide generation of surgical guides. Fornaro et al. [7] present another approach to plate design, in which a virtual plate model is deformed for planning pre-bending of physical plates. Voss et al. [24] describe a system for fracture reduction that allows design of simple surgical guides.

The previous examples use polygonal, or mesh-based, representations for the modeling. Volumetric, or voxel-based, representations can be more suitable for organic shapes such as the human anatomy, and make it easier to guarantee a 3D printable result when computing for example surface offsets. Geomagic Freeform [9] is an example of a commercial voxel-based sculpting software that allows creating patient-specific parts such as surgical guides and implants. It provides a traditional 2D UI, combined with six DOF input and haptic feedback for more intuitive 3D interaction. However, the software is too complex for most clinicians to use, as it is not aimed specifically at surgery planning.

For an overview of state-of-the-art in virtual surgery planning and implant design, see Ritacco et al. [23].

3 METHODS

Our approach is to extract a shell (Figure 2f) around the bone that serves as template for the part the surgeon wishes to design, i.e., a surgical guide or plate, from a grayscale CT (Figure 2a) and a binary segmentation (Figure 2b) of the bone, and generate a constructive solid geometry (CSG) tree that includes the shell and other components of the part. The CT and segmentation data are loaded as volume images. The shapes for the shell and other components in the CSG tree are represented as signed distance fields (SDFs) computed from the image data (Figures 2d–2e). Other inputs for generating the CSG tree include osteotomy and resection planes from the planning and user-generated inputs such as control points for defining geometry. By evaluating the CSG tree, we obtain an image from which a 3D printable triangle mesh can be extracted and, finally, exported to stereolithography (STL) mesh format.

3.1 Signed Distance Fields

A signed distance field (SDF) maps a point to a positive or negative distance scalar value, depending on whether the point lies inside or outside an object in the image. While SDFs have many applications [11], we are mainly interested in their use for implicit solid modeling. An early example of such use is found in Payne et al. [21], who describe operations such as Boolean set operations. A more recent example is the framework presented by Museth et al. [16].

3.2 Anti-Aliased Distance Transform

A distance transform (DT) is an efficient way of computing SDFs from binary images. Among approximate (non-Euclidean) 3D DTs, the Chamfer DT with integer

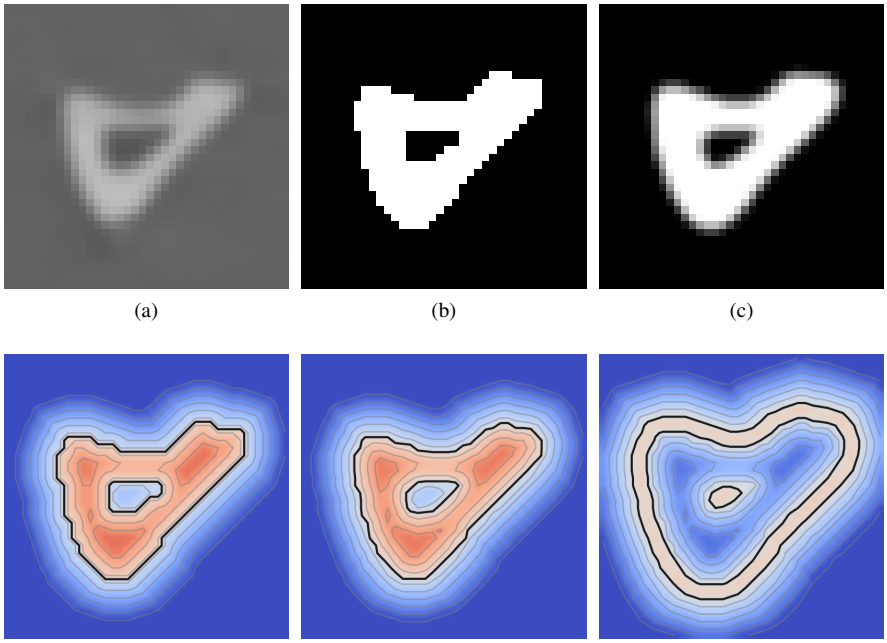


Figure 2: 2D slices of 3D image data for a fibula: (a) grayscale CT; (b) binary segmentation; (c) fuzzy coverage representation; (d) SDF from binary DT; (e) SDF from AA DT; (f) shell SDF computed from (e). In (d–f), the black iso-contour represents the zero-level, whereas the color indicate positive (red) or negative (blue) distance.

weights $\langle 3,4,5\rangle$ is fast to compute and has a bounded error of 11.8% to the Euclidean distance [2]. Efficient algorithms for computing the Euclidean DT in 3D include the vector propagation algorithm by Danielsson [5] and the more recent method by Felzenszwalb and Huttenlocher [6], both which can be parallelized for faster computation.

We implemented the Chamfer $\langle 3,4,5\rangle$ DT algorithm, since it is easy to implement, and because accurate distances are only needed close (within a few millimeters) to the bone template surface; we also aim to keep the memory requirements for computing and storing the SDFs low. We adopt the convention that voxels inside objects have positive values and voxels outside objects have negative values, and compute the SDF in two passes, one for the internal distances and one for the external distances. The computed distance values are stored as 8-bit signed integers, such that integer distances are constrained (clamped) to the range $[-128,127]$.

Anti-aliased (AA) DTs can produce more accurate distance values compared to binary DTs, in particular close to object surfaces, by using sub-pixel or sub-voxel information in grayscale image data. Other benefits are a smoother iso-surface for visualization and triangle mesh extraction, and a gradient magnitude closer to 1 near the zero-level isosurface (Figure 2e). Gustavson and Strand [10] propose an extension of Danielsson's [5] vector propagation algorithm, in which they incorporate pixel coverage information when computing the DT; they suggest a fast linear approximation d_f for mapping coverage values to sub-pixel (or sub-voxel) distances,

$$d_f = 0.5 - a, \tag{1}$$

where a is a fuzzy coverage value in the range $[0,1]$.

To retain the performance of computing the Chamfer DT, we extend our Chamfer DT implementation to incorporate voxel coverage information, instead of implementing the AA Euclidean DT in [10]. In each pass (interior and exterior), we use Equation 1 to initialise foreground voxels that otherwise would be assigned the value zero with sub-voxel distances computed from coverage, in order to compute an AA DT.

3.3 Binary-to-Coverage Conversion

CT images exhibit fuzzy tissue boundaries due to the partial volume effect (PVE), and also from filtering for removing noise. Segmented CT data, in contrast, often provide binary label masks. When the binary segmentation is computed by thresholding and a subsequent labeling step, for example using the method by Nysjö et al [17], so that foreground voxels in label masks correspond to foreground voxels in the original thresholded image, we can convert the binary label masks to fuzzy coverage representations suitable as input to an AA DT.

To perform binary-to-coverage conversion, we extract a 2-voxel thick boundary around each labeled object, containing the interior and exterior boundary voxels computed for 27-connectivity. At each boundary voxel,

(a) (b) (c) (d)

Figure 3: Fibula cutting guides: (a–b) slot type (front and back); (c–d) flange type (front and back).

we supersample the original grayscale CT image by trilinear interpolation at 4^3 (64) sub-voxel positions in a uniform grid, and compute coverage as $a = \frac{1}{N}\sum_{i=0}^{N-1} x_i$, where N is the number of samples and $x_0,\ldots,x_{N-1}$ are thresholded sample values. The resulting fuzzy coverage representation (Figure 2c) allows sub-voxel distances to be estimated at a resolution of $\frac{1}{4}$ voxel, using Equation 1.

3.4 Shell Generation

In order to generate a surgical guide or plate that fits the anatomy of the patient, we should compute a shell at an accurate and precise offset from the bone template. We generate the shell by applying the function

$$d_{shell}(x) = max(d_{inner} - x, x - d_{outer}), \qquad (2)$$

where d_{inner} and d_{outer} define the distance offsets from the original iso-surface to the inner and outer iso-surfaces of the shell, respectively, to the SDF of the bone template (Figure 2d or 2e), resulting in a new SDF for the shell (Figure 2f).

3.5 Constructive Solid Geometry

Constructive solid geometry (CSG) provides a compact representation for Boolean set operations (unions, intersections, and differences) on solid shapes, by storing shapes and operations as nodes in a binary tree. It can be used for implicit solid modeling with SDFs, by using Boolean set operators for SDFs [11] and images for storing shapes and intermediate results.

For each guide or plate model, we generate a CSG tree containing the shell and other components needed for the part, as described in the next three sections. SDFs for other components are either computed from voxelised triangle meshes or sampled from distance functions of basic primitives [22].

As a final step, when the CSG result image should only contain a single connected component, we use connected component analysis to remove smaller components, such as unwanted material generated where the bone template is hollow.

3.5.1 *Fibula Cutting Guides*

Fibula cutting guides are automatically created when the surgeon presses a button. The cut planes for the fibula osteotomies are used to cut the shell SDF of the bone template, the number of cut planes used determined by which type of guide (slots or flanges) the surgeon wishes to create (Figure 3). When generating the CSG tree, we insert components for slots and flanges, bridges for connecting the guide segments, and drill guides for the plate screw holes. We also compute the difference with the bone template SDF after adding the d_{inner} shell offset to the SDF, to remove unwanted material inside the guide. Evaluating a CSG tree takes a few seconds, which allows the surgeon to try different designs and adjust the orientation of the guide.

3.5.2 *Plates*

To create plates, the surgeon uses the stylus to place control points on the surface of the bone template, where the plate should be in contact with the bone (Figure 4a), and also uses the stylus to place markers for screws (Figure 4b). The plate is updated at interactive speed, and existing control points and screw markers may be moved or deleted with the stylus. The SDF of the bone template is also used as an aid to snap control points to the surface and automatically orient screw markers in the SDF gradient direction. When generating the CSG tree, we insert spheres interpolating the control points, and holes at the screw markers. We also compute the intersection with the shell SDF from the bone template, to generate the final plate (Figure 4c).

3.5.3 *Mandibular Resection Guides*

A 3D brush (Figure 5a) is used to define the geometry of the mandibular resection guides (Figure 5b). The surgeon controls the brush with the stylus, and switches between additive and subtractive brushes. When generating the CSG tree, we insert spheres for brush strokes. We also compute the intersection with the shell SDF from the bone template, and insert drill guides for the plate screw holes (Figure 5c).

(a) (b) (c)

Figure 4: Plate design: (a) plate geometry (in blue) interpolated along control points the user places on the bone template (in green) using the stylus; (b) markers for screws; (c) final plate with screw holes inserted.

(a) (b) (c)

Figure 5: Mandibular resection guide design: (a) 3D brush tool; (b) painted guide geometry; (c) final resection guide with flanges and drill guides inserted.

3.6 Visualization and Haptic Rendering

We use Marching cubes [13] (MC) to visualize the iso-surfaces of the bone templates and the models from the generated CSG trees. For the CSG trees, we utilize a sparse block data structure to keep track of blocks that contain modified data and update the iso-surface only in those blocks during modeling. The MC iso-surfaces are also exported to STL mesh format for 3D printing.

For the haptic rendering, we convert bone fragments into so called Voxmap Pointshell [15] representations. This enables haptic interaction between bone surfaces and the stylus, as well as haptic interaction between bone fragments. Further details about the haptic rendering are provided in [18].

3.7 Implementation Details

We implement the tools for surgical guide and plate generation in the HASP software, using C++ and our own OpenGL 3.2 (Core profile) based rendering framework. OpenMP and single instruction/multiple data (SIMD) intrinsics are used to speed up the image processing and the CSG operations. Our MC implementation is based on the code available at [3].

4 EXPERIMENTS AND RESULTS

To evaluate our method, we performed simulated surgery planning on three cases. Each case included CT or CT angio (CTA) images, in DICOM format, of the head and neck region and of the leg from which the bone transplant was to be harvested. The test datasets are summarized in Table 1. The dataset for Case 1 consisted of the MANIX and OBELIX images from the public OsiriX DICOM repository [20]; the OBELIX image was cropped to include only the left leg. The datasets for Cases 2 and 3 consisted of images from actual patients that were about to undergo reconstructive surgery at the time of the imaging.

We used the BoneSplit [17] software to convert the DICOM image stacks to volume images in VTK format and to segment the mandibles from the head-neck CT images and the fibulas from the leg CT images. For bone thresholding, we used threshold values in the range $[146, 261]$ Hounsfield units (HU), corresponding to conservative values for bone. We obtained cut planes for fibula osteotomies and mandible resections from the planning software, and also computed fuzzy coverage representations of the reconstructed mandibles. Images with anisotropic voxels were resampled to isotropic voxels before we computed the SDFs. After consulta-

CT	Size (voxels)	Voxel spacing (mm)
Head 1	$512 \times 512 \times 460$	$0.49 \times 0.49 \times 0.70$
Leg 1	$225 \times 241 \times 443$	$0.74 \times 0.74 \times 1.00$
Head 2	$512 \times 512 \times 159$	$0.30 \times 0.30 \times 1.00$
Leg 2	$137 \times 141 \times 135$	$0.88 \times 0.88 \times 3.00$
Head 3	$512 \times 512 \times 409$	$0.40 \times 0.40 \times 0.60$
Leg 3	$157 \times 149 \times 613$	$0.63 \times 0.63 \times 0.70$

Table 1: Test datasets.

Model	Shell offset (mm) (specified)	Min. distance (mm) with Binary DT	Min. distance (mm) with AA DT
Fibula cutting guide 1	2.0	1.26	1.57
Mandibular resection guide 1	0.5	-	0.26
Plate 1	0.5	0.06	0.17
Fibula cutting guide 2	2.0	1.15	1.51
Mandibular resection guide 2	0.5	-	0.04
Plate 2	0.5	0.26	0.30
Fibula cutting guide 3	2.0	1.41	1.63
Mandibular resection guide 3	0.5	-	0.25
Plate 3	0.5	0.21	0.28

Table 2: Distance measurements.

tion with clinicians, we decided to use a shell offset of 2.0 mm for the fibula cutting guides, to take the membrane around the bone of the fibula into account, and a shell offset of 0.5 mm for the plates and mandibular resection guides, to provide some tolerance for the choice of bone threshold value in the segmentation.

To quantitatively evaluate the accuracy and precision of the generated models, we exported STL files of iso-surfaces of bone templates and models and imported the files in ParaView [1]. We used a Hausdorf distance filter plugin for ParaView [4] to compute surface-to-surface distances between bone templates and models (Figure 6). Fibula cutting guides and plates were generated with binary and AA DTs (Figure 7), wheras mandibular resection guides had similar properties (same shell offset) as plates and were generated only with AA DT. Table 2 compare the expected shell offset with the minimum distance computed for each object. Using the AA DT resulted in minimum distances closer to the expected shell offsets, and visibly smoother iso-surfaces, compared to the binary DT. The low minimum distance for the mandibular resection guide in Case 2 was caused by two drill guide components being inserted too close to the bone.

To test the manufacturability, we also printed a number of exported STL files in polyactide (PLA) plastic on an Ultimaker Original 3D printer (Figure 8).

The experiments were performed on a laptop with an Intel Core i7-4710MQ CPU, 16 GB of RAM, and a NVIDIA Quadro K4100M GPU.

5 DISCUSSION

Surgical guide and plate models can be efficiently generated in a few minutes from segmented CT images with the methods we present in this paper. We are currently preparing a validation study in which we for a larger number of patient cases will compare the planned outcome from our software with the actual outcome of simulated surgery on 3D printed plastic bones and models created from the plan.

An issue of the voxel-based modeling is the limited resolution it provides. Using an anti-aliased DT allows us to preserve sub-voxel distances, when fuzzy coverage representations are available, but fine details such as threads for screw holes require higher resolution. Adaptive distance fields [8], or performing parts of the CSG operations on the final iso-surface mesh, could be viable options to explore.

The use of haptics is also something to explore further for this type of application. In particular, haptic feedback for testing the fit of models during the design could provide valuable information to the user, and will eventually be added to our software.

ACKNOWLEDGEMENT

We like to thank Johan Nysjö, Centre for Image Analysis, for providing access to the BoneSplit segmentation software. We would also like to thank Andreas Thor, Andrés Rodríguez-Lorenzo, and Jan-Michaél Hirsch from the Department of Surgical Sciences, Uppsala University, as well as Daniel Buchbinder from the Mount Sinai School of Medicine, NY, for their clinical input and for providing datasets for Cases 2 and 3.

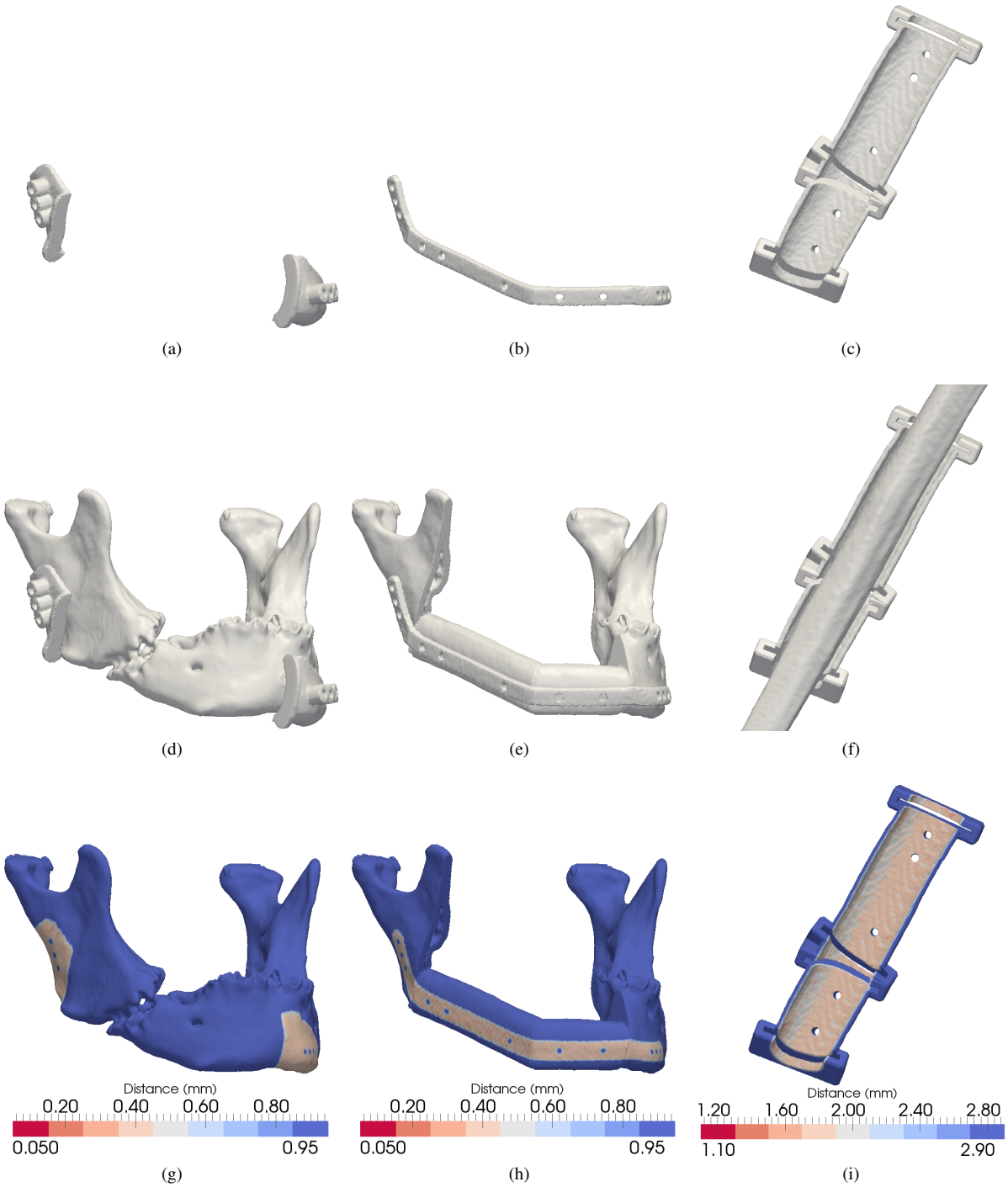
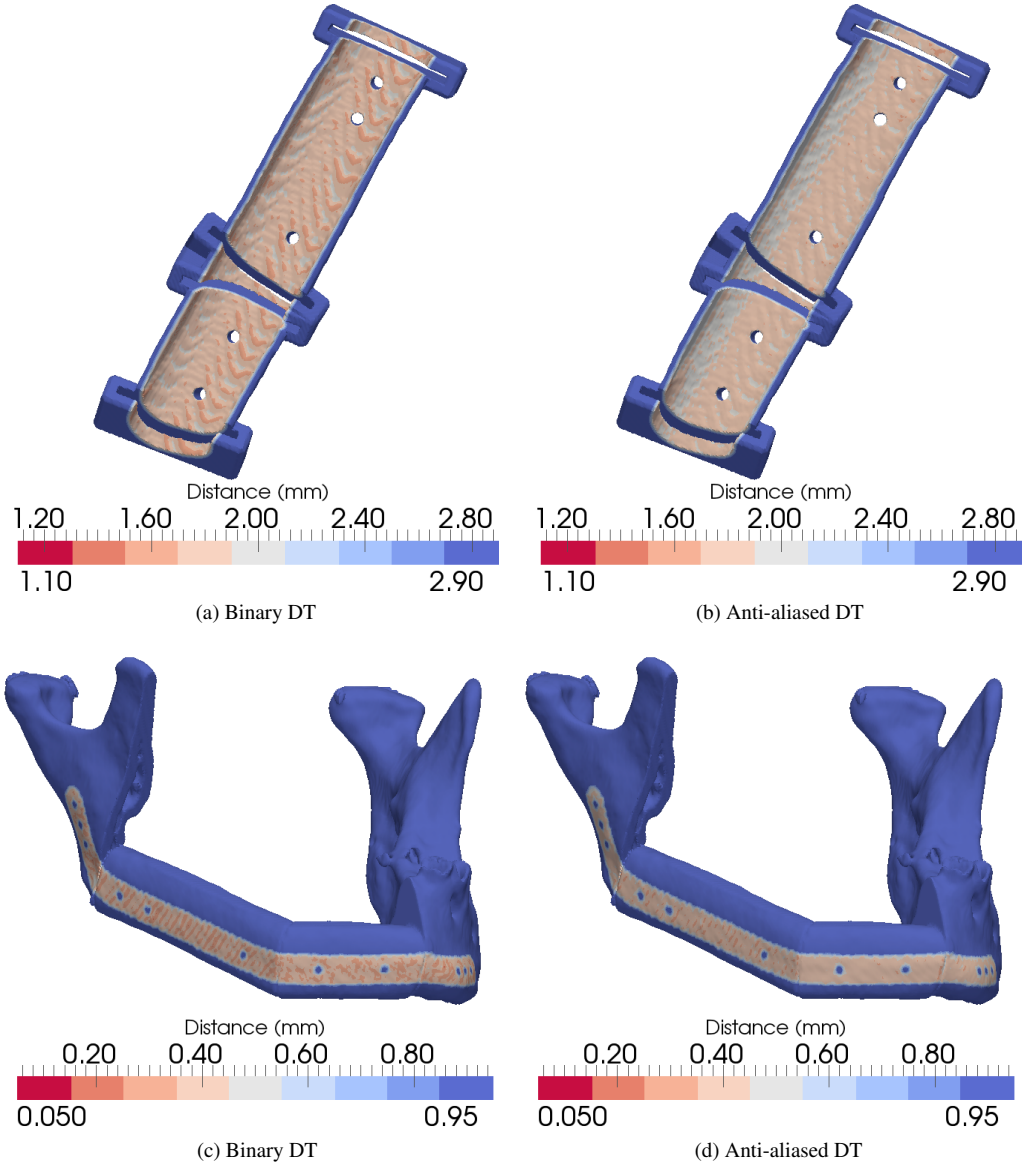


Figure 6: Results for Case 3: top row (a–c): iso-surfaces of generated guide and plate models; middle row (d–f): iso-surfaces of generated models with bone templates; bottom row (g–i): computed bone-to-model distances.



(a) Binary DT (b) Anti-aliased DT

(c) Binary DT (d) Anti-aliased DT

Figure 7: Comparison of bone-to-model distances for models generated with binary and anti-aliased DTs: (a–b) fibula cutting guide; (c–d) plate. This figure is best viewed on a monitor.

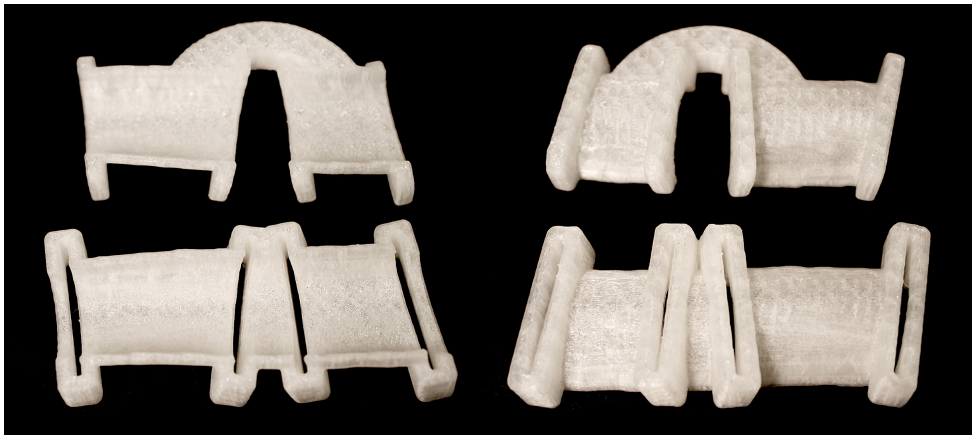


Figure 8: Fibula cutting guide models 3D printed in polyactide (PLA) plastic.

6 REFERENCES

[1] AHRENS, J., GEVECI, B., AND LAW, C. ParaView: An End-User Tool for Large Data Visualization, Visualization Handbook. Elsevier, 2005, ISBN-13: 978-0123875822.

[2] BORGEFORS, G. Distance Transformations in Arbitrary Dimensions. *Computer Vision, Graphics, and Image Processing 27*, 3 (1984), 321–345.

[3] BOURKE, P. Polygonising a scalar field. `http://paulbourke.net/geometry/polygonise`. Accessed on February 9, 2017.

[4] COMMANDEUR, F., VELUT, J., AND ACOSTA, O. A VTK Algorithm for the Computation of the Hausdorff Distance. *The VTK Journal* (2011). `http://hdl.handle.net/10380/3322`.

[5] DANIELSSON, P.-E. Euclidean Distance Mapping. *Computer Graphics and Image Processing 14*, 3 (1980), 227–248.

[6] FELZENSZWALB, P. F., AND HUTTENLOCHER, D. P. Distance Transforms of Sampled Functions. *Theory of Computing 8* (2012), 415–428.

[7] FORNARO, J., KEEL, M., HARDERS, M., MARINCEK, B., SZEKELY, G., AND FRAUENFELDER, T. An interactive surgical planning tool for acetabular fractures: initial results. *Journal of Orthopaedic Surgery and Research 50*, 5 (2010).

[8] FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2000), SIGGRAPH '00, ACM, pp. 249–254.

[9] GEOMAGIC. Freeform. `http://www.geomagic.com/en/products/freeform/overview`. Accessed on February 9, 2017.

[10] GUSTAVSON, S., AND STRAND, R. Anti-aliased Euclidean distance transform. *Pattern Recognition Letters 32*, 2 (2011), 252–257.

[11] JONES, M. W., BAERENTZEN, J. A., AND SRAMEK, M. 3D Distance Fields: A Survey of Techniques and Applications. *IEEE Transactions on Visualization and Computer Graphics 12*, 4 (2006), 581–599.

[12] KOVLER, I., JOSKOWICZ, L., WEIL, Y., KHOURY, A., KRONMAN, A., MOSHEIFF, R., LIEBERGALL, M., AND SALAVARRIETA, J. Haptic computer-assisted patient-specific preoperative planning for orthopedic fractures surgery. *Int. Journal of Computer Assisted Radiology and Surgery 10*, 10 (2015), 1535–1546.

[13] LORENSEN, W. E., AND CLINE, H. E. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics 21*, 4 (1987), 163–169.

[14] MATERIALISE. Mimics. `http://www.materialise.com/en/medical/mimics-innovation-suite`. Accessed on February 9, 2017.

[15] MCNEELY, W. A., PUTERBAUGH, K. D., AND TROY, J. J. Six Degree-of-freedom Haptic Rendering Using Voxel Sampling. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1999), SIGGRAPH '99, ACM, pp. 401–408.

[16] MUSETH, K., BREEN, D. E., WHITAKER, R. T., AND BARR, A. H. Level set surface editing operators. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 330–338.

[17] NYSJÖ, J., MALMBERG, F., SINTORN, I.-M., AND NYSTRÖM, I. BoneSplit - A 3D Texture Painting Tool for Interactive Bone Separation in CT Images. *Journal of WSCG 23*, 2 (2015), 157–166.

[18] OLSSON, P., NYSJÖ, F., HIRSCH, J.-M., AND CARLBOM, I. B. A Haptics-Assisted Cranio-Maxillofacial Surgery Planning System for Restoring Skeletal Anatomy in Complex Trauma Cases. *Int. Journal of Computer Assisted Radiology and Surgery 8*, 6 (2013), 887–894.

[19] OLSSON, P., NYSJÖ, F., RODRIGUEZ-LORENZO, A., THOR, A., HIRSCH, J.-M., AND CARLBOM, I. B. Haptics-assisted Virtual Planning of Bone, Soft Tissue, and Vessels in Fibula Osteocutaneous Free Flaps. *Plastic and Reconstructive Surgery - Global Open 3*, 8 (2015).

[20] OSIRIX. DICOM Image Library. `http://www.osirix-viewer.com/resources/dicom-image-library`. Accessed on February 9, 2017.

[21] PAYNE, B. A., AND TOGA, A. W. Distance field manipulation of surface models. *IEEE Computer Graphics and Applications 12*, 1 (1992), 65–71.

[22] QUILEZ, I. Distance functions. `http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm`. Accessed on February 9, 2017.

[23] RITACCO, L. E., MILANO, F. E., AND CHAO, E. *Computer-Assisted Musculoskeletal Surgery: Thinking and Executing in 3D*. Springer International Publishing, 2016.

[24] VOSS, J. O., VARJAS, V., RAGUSE, J.-D., THIEME, N., RICHARDS, R. G., AND KAMER, L. Computed tomography-based virtual fracture reduction techniques in bimandibular fractures. *Int. Journal of Cranio-Maxillofacial Surgery 44*, 2 (2015), 177–185.

[25] ZWEIFEL, D. F., SIMON, C., HOARAU, R., PASCHE, P., AND BROOME, M. Are Virtual Planning and Guided Surgery for Head and Neck Reconstruction Economically Viable? *Journal of Oral and Maxillofacial Surgery 73*, 1 (2015), 170–175.

Least Squares Affine Transitions for Global Parameterization

Ana Maria Vintescu
LTCI - Télécom ParisTech - Institut Mines-Telecom
75013 Paris, France
vintescu@telecom-paristech.fr

Florent Dupont
Université de Lyon, CNRS
Université Lyon 1, LIRIS UMR 5205
69622 Villeurbanne, France
florent.dupont@liris.cnrs.fr

Guillaume Lavoué
Université de Lyon, CNRS
INSA-Lyon, LIRIS UMR 5205
69621 Villeurbanne, France
glavoue@liris.cnrs.fr

Pooran Memari
LIX UMR 7161, CNRS, École Polytechnique, Université Paris Saclay
91128 Palaiseau Cedex - France
memari@lix.polytechnique.fr

Julien Tierny
Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606
75005 Paris, France
julien.tierny@lip6.fr

ABSTRACT

This paper presents an efficient algorithm for a global parameterization of triangular surface meshes. In contrast to previous techniques which achieve global parameterization through the optimization of non-linear systems of equations, our algorithm is solely based on solving at most two linear equation systems, in the least square sense. Therefore, in terms of running time the unfolding procedure is highly efficient. Our approach is direct – it solves for the planar UV coordinates of each vertex directly – hence avoiding any numerically challenging planar reconstruction in a post-process. This results in a robust unfolding algorithm. Curvature prescription for user-provided cone singularities can either be specified manually, or suggested automatically by our approach. Experiments on a variety of surface meshes demonstrate the runtime efficiency of our algorithm and the quality of its unfolding. To demonstrate the utility and versatility of our approach, we apply it to seamless texturing. The proposed algorithm is computationally efficient, robust and results in a parameterization with acceptable metric distortion.

0.1 Keywords

surface parameterization, geometry processing,triangular mesh, mesh unfolding

1 INTRODUCTION

Surface parameterization represents a main topic in geometry processing and computer graphics fields. It is defined as a one-to-one mapping between a surface and typically a 2D plane, where geometrical tasks can be carried out more efficiently. The most important application of surface parameterization are texture mapping, texture synthesis, re-meshing, and morphing. In order to unfold a surface to the plane, it must have a disk topology; for a closed surface this requirement implies cutting it into a single or multiple disk topology charts. Cutting can result in visual artifacts due to the discontinuities across the boundaries of the charts. To this extent, methods for *global parameterization* of triangulated surfaces have been proposed. Within this framework, the global parameterization of a surface with disk topology can be defined as a homeomorphism from the surface to a subset of the plane, such that the discrete Gaussian curvature, i.e. the difference between 2π and the incident triangles' sum of angles at a vertex, is zero everywhere except for a few vertices called *cone singularities*. These can be thought of distortion absorbers, being chosen as vertices of the mesh where large area distortion can be predicted prior to the actual parameterization, [Kha05].

Several approaches based on metric scaling have been proposed in the past to address global parameterization [Jin08, Yan09]. However, these methods mostly rely on non-linear solvers and are hence computationally expensive. Linearized approximations, although computationally attractive, are imprecise (the target metric is only approximated and therefore is not guaranteed to be flat). More importantly, the planar coordinates of the surface vertices (the actual output) are not the variables that are optimized by this family of techniques. We will refer to those as *indirect methods* [Ben08]. Indeed

they focus on the surface metric, i.e. the edge lengths, and later reconstruct the planar coordinates in a post-process. However, this reconstruction post-process may be computationally expensive and, more importantly, numerically challenging. This paper addresses these two issues by presenting a global parameterization technique which is fast by employing linear solvers, which minimizes angular distortion through imposed conformality, and reduces the area distortion through the use of cone singularities. Our method is *simple* and *direct*. It directly solves for the 2D coordinates. Thus, it does not suffer from numerical instabilities due to angle-to-uv or scaling factors-to-uv conversions, as found with indirect approaches, [She06]. In contrast to more computationally expensive techniques based on non-linear solvers, the computational speed of our approach makes it a good candidate for interactive applications, such as user-driven parameterization improvement for instance, where the users could interactively adjust the number and locations of the cones.

Contributions

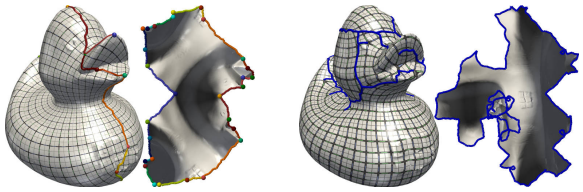
This paper makes the following new contributions:

1. **A fast and robust global parameterization algorithm:** Our method is direct (hence robust), non-iterative and only relies on the solving of at most two linear systems.
2. **Automatic curvature prescription:** Given a list of cone singularities, we present a fast algorithm to automatically evaluate relevant curvature prescriptions at the cone singularities.

The next section presents related work. Next, we introduce the method and its preliminaries. Sec. 4 and 5 present the proposed global parameterization algorithm in detail. The modeling of the linear systems is described in Sec. 6, while experimental results are reported in Sec. 7. To demonstrate the utility and versatility of our technique, we present its application to seamless texturing in Sec. 8 and finally, Sec. 9 concludes the paper.

2 RELATED WORK

In the following, we will only focus on surface parameterization techniques that are related to our work. We refer the reader to survey articles [Flo05, She06] for further reading. Most existing parameterization methods focus on *conformal* parameterizations (where angle distortion is minimized). Several methods [Des02, Lev02, Liu08, Ray03] focus on parameterizing surfaces of disk topology while reducing angular distortion. These methods employ linear solvers for the minimization of energy functions (that are discrete analogous to Laplace and Cauchy-Riemann equations) defined in terms of the 2D coordinates of the vertices in the mesh. These are therefore *direct* methods. They allow a free boundary setting but pin two vertices to avoid a non-trivial solution (a more recent approach removes such necessity through a spectral embedding [Mul08]).



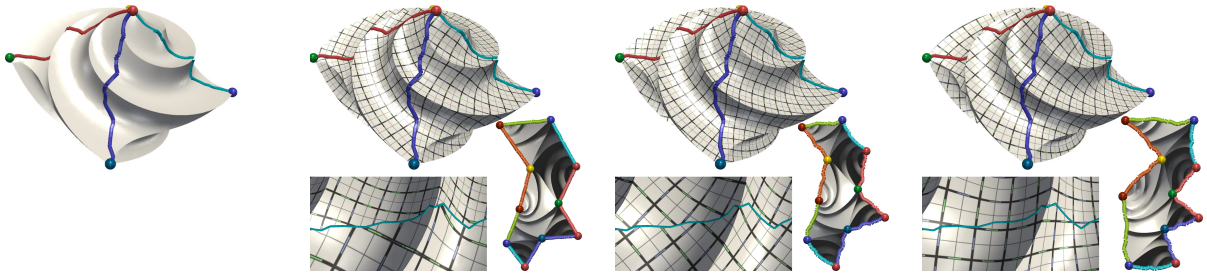
(a) *LSGP* (b) *MIQ*

Figure 1: Comparison between our approach (a) and Mixed-Integer Quadrangulation (MIQ) [Bom09] (b). On this example, the MIQ approach generates many boundary self-intersections, see the unfolded blue boundary.

Indirect methods [She05, She00, Zay07] aim at minimizing the difference between the initial angles of the 3D mesh and the final ones. The methods are computationally expensive (for advances see linearized version [Zay07]) and suffer from numerical instability when converting the obtained angles to actual 2D coordinates.

Indirect global parameterization methods [Ben08, Spr08, Kha05] determine the necessary edge lengths to parameterize the mesh to the plane before cutting it along a set of cut-paths to disk topology. Moreover, they also make use of cone singularities to absorb the curvature (i.e. the angle deficits), resulting in a global parameterization where the scaling of the surface is continuous across each cut-path.

While expensive non-linear solvers are usually employed [Jin08, Kha06, Spr08, Yan09], Ben Chen et al. [Ben08] approximate the solution through a Finite Element discretization of the Poisson equation, yielding better computational complexity at the expense of metric accuracy. Some methods [Spr08, Myl12, Myl13] additionally provide the possibility of obtaining seamless parameterizations by iteratively quantizing the cone angle deficits to multiples of $\pi/2$ and rectifying the cone positions to integer locations. Myles and Zorin [Myl12] compute seamless parameterizations, by employing linear solvers with linear constraints in an iterative manner for the first two steps of their algorithm (cone detection and curvature prescription). However, the last step consists in optimizing the non-linear as-rigid-as-possible (ARAP) energy function. Even though the first two steps solve linear systems, they do so in an iterative fashion and many iterations may be required. Moreover, the last step still requires the use of a time-consuming non-linear solver. Quadrangulation techniques based on structure-aligned parameterizations [Ray06, Ton06, Kal07, Bom09, Cam15, Myl14] are also related to global parameterization (for a more



(a) (b) (c) (d)

Figure 2: Algorithm overview: (a) Given an input triangular mesh and cone singularities, the mesh is cut through the cones in order to obtain a disk topology. The cones are shown by colored spheres and conic cuts by colored cylinders. (b) Next, the mesh is conformally parameterized by concentrating the entire curvature at the cones. (c) Given the cone angles resulting from the previous step, the surface is globally parameterized. (d) Optionally, the mesh can be seamlessly parameterized.

detailed description see [Bom12]). Bommes et al. [Bom09] obtain quadrangulations by solving two mixed integer problems, one for the computation of a direction-aware cross field [Ray08, Pan12, Kno13] and one for the global parameterization, with additional similar solves in case of singularity relocation. The transition functions across cut-paths that we employ are similar to the ones used by Bommes et al. [Bom09] and Myles et al. [Myl12]. However, we formulate such constraints in a different optimization setting, which is based on faster, linear solvers.

Although the Mixed-Integer Quadrangulation (MIQ) method [Bom09] generates high quality output quadrangulations, it can yield many boundary self-intersections in the planar domain, Fig. 1 - parameterization using the MIQ method obtained by Ebke et al. [Ebk13], which may challenge their systematic usage for sub-sequent applications (such as surface cross parameterization for instance). An extension of (MIQ) [Bom09] has been proposed to address this boundary domain intersection problem [Bom13] but at the expense of increasing further computation times.

3 METHOD OVERVIEW

3.1 Preliminaries

The input surface M is given as a mesh made of vertices (V), edges (E) and triangles (T). Their number is noted with $|V|$, $|E|$ and $|T|$ respectively. The geometry of M is given as the 3D coordinates of the vertices $X_v = (v_x, v_y, v_z), \forall v \in V$. The output parameterization is represented with 2D coordinates for each vertex $U_v = (u, v)$. The length of an edge is given by $e_{i,j} = \|X_{v_i} - X_{v_j}\|_2$ in 3D or $e_{i,j} = \|U_{v_i} - U_{v_j}\|_2$ in 2D. An angle in a triangle t is given by: $\alpha^t_{v_i} = \arccos\left(\frac{e^2_{i,j}+e^2_{i,k}-e^2_{j,k}}{2e_{i,j}e_{i,k}}\right)$, where v_i, v_j and v_k are the vertices of t.

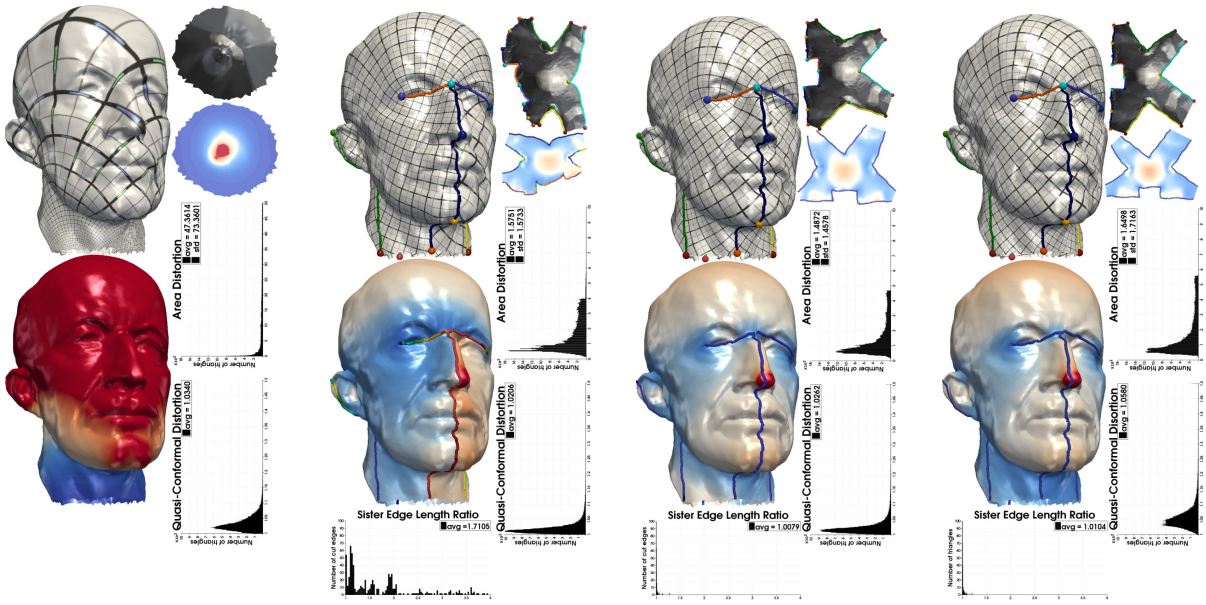
The discrete Gaussian curvature is given by $K = \left\{ k_{v_i} = \begin{cases} 2\pi - \sum_{t\in T_{v_i}}(\alpha^t_{v_i}), \text{for an interior vertex} \\ \pi - \sum_{t\in T_{v_i}}(\alpha^t_{v_i})\}, \text{for a boundary vertex} \end{cases} \right\}$, where T_{v_i} represents the set of incident triangles to the vertex v_i.

The *Gauss-Bonnet* Theorem states that the integral of the curvature is a constant, which depends on the topology of M: $\sum K = 2\pi\chi$, where χ represents the Euler characteristic of M ($\chi = |V| - |E| + |T|$). Given a mesh with disk topology, a *global* parameterization is a homeomorphism to a subset of the plane, such that the discrete Gaussian curvature is zero everywhere except at a set of selected vertices C, called *cones*.

3.2 Algorithm Description

Given an input triangular mesh, our algorithm first cuts the mesh open through a set of cut-paths that connect cone singularities. We call those *conic cuts*. Such singularities will *absorb* the area distortion of the parameterization, as showcased in Fig. 3. The second step consists in parameterizing the mesh, while minimizing angular distortion and imposing zero curvature everywhere except at the cones. This is achieved by introducing straightness conditions for the entire boundary, Fig. 2(b). By employing only conformality and boundary straightness conditions, the two sides of a conic cut might have different lengths in the plane, Fig. 2(b). To ensure that the two resulting sides of a conic cut are scaled similarly, we additionally enforce rotations and translations between each side of a conic cut, Fig. 2(c) note the continuity of the distortion across the cuts.

The rotation angles are either provided by the user or detected as the angles between the conic cuts in the parameterization that resulted from the previous step, i.e. the curvature prescription..



(a) *LSCM [Lev02]* (b) *LSCM [Lev02], conic cuts* (c) *Our approach* (d) *Our approach - Seamless*

Figure 3: Comparison between Least-Squares Conformal Maps (LSCM) [Lev02] without (a) and with conic cuts (b) and our approach (c), (d). From top to bottom: *(i)* textured surface with cones (colored spheres) and cuts (colored cylinders), as well as planar unfolding, *(ii)* area distortion (color map: blue (0) to red (10)) and sister edge length distortion on the cuts (rainbow color map), *(iii)* histograms of sister edge length, quasi-conformal and area distortions (ideal values: 1). Introducing cones (b) drastically reduces area distortion, while minimizing sister edge length distortion (c) yields a global parameterization . By imposing positional and rotational constraints on cones, and respectively on conic cuts, our approach can be used to generate seamless parameterizations (d).

4 GLOBAL PARAMETERIZATION WITH ROTATIONAL CONSTRAINTS

Given a set of cone singularities and their corresponding curvature prescriptions (either provided by the user or computed automatically, see Sec. 5), a global parameterization can be defined as an angle preserving homeomorphism from the input triangular surface to a subset of a plane, such that the discrete curvature is zero everywhere except at the cones. In this section, we present an algorithm that computes such a mapping, by minimizing angular distortion and penalizing the deviation from the target curvature in the least-squares sense. First, the surface is cut open along *conic cuts* to become homeomorphic to a disk. Second, the surface is unfolded and the target curvature is enforced by imposing affine transition functions across conic cuts.

4.1 Mesh Cutting

To be unfolded to the plane, we require the input surface to have a disk topology. For surfaces with a sphere topology, this can be obtained by introducing a boundary component, by cutting the mesh along the shortest paths that connect the cones. We detail this process hereafter. Variants of this strategy can be derived for surfaces with different genus.

The shortest path between each possible pair of cones is first computed with Dijkstra's algorithm. Next, a minimum spanning tree is constructed on a graph where the nodes denote the cones and where each edge is weighted by the geodesic distance between its cones (i.e. the length of their shortest paths). The edges of the spanning tree then correspond to the shortest paths along which the surface is cut open and that we call *conic cuts*. The valence of a cone corresponds to the number of conic cuts incident to it. The actual cutting process involves the duplication of all the surface edges found on the paths. Given an edge initially present on a shortest path, its copy is called its *sister edge*. Similarly, the copy of a conic cut is called its *sister conic cut*. Throughout the paper, sister conic cuts will be represented by curves with matching colors (see Fig. 2 for instance). Note that after the cutting, a cone may have a high valence. Also, similarly to Springborn et al. [Spr08], each boundary component is treated as a cone.

4.2 Least-Squares Conformal Maps with Rotational Constraints

Once the mesh is cut open, we unfold it to the plane with a new algorithm that minimizes angular distortion and penalizes the deviation from the target curvature.

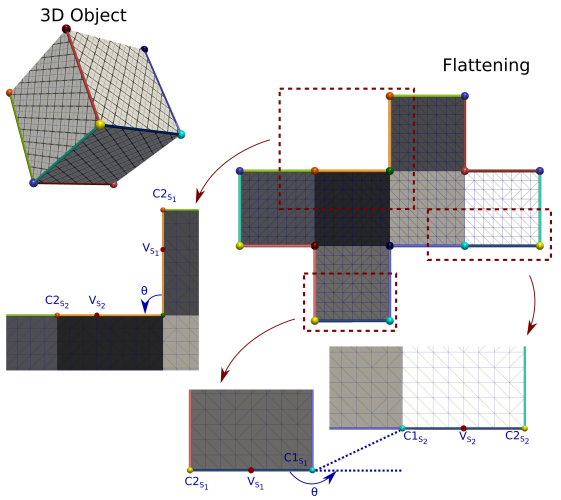


Figure 4: Rotational transformation relations between simple and complex sister conic cuts of a cube.

Our approach relies on the conformality criterion introduced by Lévy et al. [Lev02], as it enables a direct and fast optimization. We impose similar conditions as Aigerman et al. [Aig15], but in contrast to them we are not restricted to four cone configurations, allowing a general framework. Given some prescribed target curvatures for each cone, our approach consists in imposing this angle deficit by constraining combinations of translations and rotations between sister conic cuts, which is achieved as follows. Sister conic cuts can be classified into two categories:

1. *Simple conic cuts* - cuts connected to a cone of valence 1 (see the orange cuts connected to the green cone of valence 1 in Fig. 4);
2. *Complex conic cuts* - cuts not connected to a cone of valence 1 (see the dark blue cut connected in Fig. 4).

Note that simple sister cuts will be adjacent in the plane while complex ones will not. Given a cone of valence 1, we enforce its prescribed angle deficit θ by constraining its incident sister cuts to be related by a rotation of angle θ (Fig. 4, left inset zoom). For complex sister cuts, we first translate them to the origin (translation $T2$), apply the required rotation of angle θ (rotation R) and translate them back to their original location (translation $T1$):

$$\begin{bmatrix} V_{s_2u} \\ V_{s_2v} \\ 1 \end{bmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & C1_{s_2u} \\ 0 & 1 & C1_{s_2v} \\ 0 & 0 & 1 \end{pmatrix}}_{T1} \cdot \underbrace{\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{R} \cdot \underbrace{\begin{pmatrix} 1 & 0 & -C1_{s_1u} \\ 0 & 1 & -C1_{s_1v} \\ 0 & 0 & 1 \end{pmatrix}}_{T2} \begin{bmatrix} V_{s_1u} \\ V_{s_1v} \\ 1 \end{bmatrix} \tag{1}$$

where (V_{s_1u}, V_{s_1v}) and (V_{s_2u}, V_{s_2v}) are the unknown (u,v) coordinates of the cut vertex V_{s_1} and its sister cut vertex V_{s_2} (Fig. 4), and where $(C1_{s_1u}, C1_{s_1v})$ and $(C1_{s_2u}, C1_{s_2v})$ are the unknown (u,v) coordinates of the cone $C1$ on the conic cut s_1 and of its duplicate on the sister conic cut s_2 (Fig. 4). For simple conic cuts, the latter two cones will coincide (the cone being of valence one). Therefore, for each vertex along a conic cut, we add the following two equations to the least-squares conformal map system:

$$\begin{aligned} V_{s_2u} &= V_{s_1u} \cdot \cos(\theta) - V_{s_1v} \cdot \sin(\theta) - C1_{s_1u} \cdot \cos(\theta) \\ &\quad + C1_{s_1v} \cdot \sin(\theta) + C1_{s_2u} \\ V_{s_2v} &= V_{s_1u} \cdot \sin(\theta) + V_{s_1v} \cdot \cos(\theta) - C1_{s_1u} \cdot \sin(\theta) \\ &\quad - C1_{s_1v} \cdot \cos(\theta) + C1_{s_2v} \end{aligned} \tag{2}$$

5 CURVATURE PRESCRIPTION ESTIMATION BY STRAIGHTNESS CONSTRAINTS

So far, we assumed that the target curvatures, i.e. the corresponding target θ angles, were provided by the user. We describe in this section a new, fast algorithm for the automatic evaluation of relevant curvature prescriptions for a set of input cone singularities. The cone singularities can be either user-provided or automatically extracted with an existing technique ([Ben08, Spr08, Myl12]). The key idea of our algorithm is to unfold the input surface while minimizing in the least-squares sense angle distortion as well as the deviation from zero curvature, everywhere except at the cones. With this strategy, cone angles will self-adjust to provide a good balance between cone curvature absorption and angular distortion. This procedure can be interpreted as a redistribution of the surface curvature onto the cones in a least-squares sense. As described below, an appealing aspect of this method is that it only requires a single linear solving. Hence, it is very efficient in terms of computation time. To penalize the deviation from zero curvature, we enforce straightness constraints. First, for each vertex located on a conic cut, we evaluate its (3D) arc-length parameterization along the conic cut. This parameterization will be used as barycentric coordinates to enforce the alignment of the conic cut in 2D, as follows. We denote the original edge lengths vectors along a conic cut as $L_{CP} = [e_{V_{C_1},V_s^1}, ..., e_{V_s^i,V_s^{i+1}}, ..]$, Fig. 5, the total conic cut lengths as $L_{CP}^{Tot} = \sum_{e \in CP} L_{CP}[e]$; the relative edge lengths will be: $r_{CP}[e] = [\frac{L_{CP}[e]}{L_{CP}^{Tot}}, \forall e \in CP$.

We calculate the cumulative sum for the ratio vector and obtain: $R_{CP} = [0, r_{CP}^1, r_{CP}^1 + r_{CP}^2, ..., 1]$.

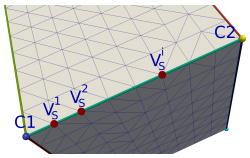


Figure 5: Conic cut.

To enforce the straightness of the conic cuts in the plane, we impose that each vertex along a cut is placed at a location which is dictated by the linear interpolation between the ending points of the cut, with factors r^i_{CP}. This yields two new equations that we add to the least-squares conformal map system:

$$\begin{aligned} V^i_{s\,u} &= C1_u \cdot (1 - r^i_{CP}) + C2_u \cdot (r^i_{CP}) \\ V^i_{s\,v} &= C1_v \cdot (1 - r^i_{CP}) + C2_v \cdot (r^i_{CP}) \end{aligned} \tag{3}$$

where $(V^i_{s\,u}, V^i_{s\,v})$, $(C1_u, C1_v)$ and $(C2_u, C2_v)$ stand for the unknown (u,v) coordinates of the cut vertex V_s and the cones $C1$ and $C2$ respectively, Fig. 5. The result of this least-squares solution is illustrated in Fig. 2(b), where the conic cuts have been straightened in the plane. From there, our algorithm collects for each cone

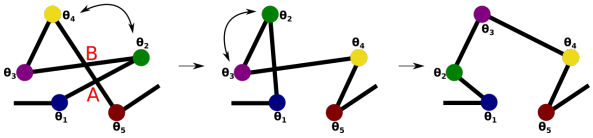


Figure 6: Possible cut intersections are recursively resolved by cone swapping. Here, the intersection A between segments $[\theta_1, \theta_2]$ and $[\theta_4, \theta_5]$ is first solved instead of intersection B because it counts the most intersections between its two extremities θ_1 and θ_5.

the angle between its incident conic cuts and provides this value as a curvature prescription for the rest of the pipeline, Fig. 2(c). This approach is non-iterative and simple to implement, requiring only solving one linear system. By applying these straightness conditions for all paths, it is possible that such paths intersect. If such a configuration is encountered, we simply recursively swap the positions of their extremity cones. In particular, this swapping procedure processes intersections in decreasing order of the number of remaining intersections between their path extremities, Fig. 6.

6 FORMULATING THE LINEAR SYSTEM

In this section, we detail how the equations discussed in the previous sections can be integrated in the least-squares conformal map system. The least-squares conformal map (LSCM) method [Lev02] defines the conformality of the mapping in terms of its gradients: the gradient vectors inside a triangle should be orthogonal and have the same norm. Thus, the authors propose the minimization of the following energy E_{LSCM}:

$$E_{LSCM} = \sum_{T_j \subset T} A_{T_j} \left\| \nabla v - \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \nabla u \right\|^2 \tag{4}$$

where A_{T_j} represents the area of a triangle T_j and ∇u and ∇v stand for the gradient of the (u,v) coordinates within the triangle $T_j = \{p_1, p_2, p_3\}$:

$$\begin{aligned} \nabla u = (&X_{p_1} \cdot (v_{p_2} - v_{p_3}) + X_{p_2} \cdot (v_{p_3} - v_{p_1}) \\ &+ X_{p_3} \cdot (v_{p_1} - v_{p_2}))/(2 \cdot A_{T_j}) \end{aligned} \tag{5}$$

Similarly, for ∇v. In order to obtain a non-trivial solution, the authors fix the 2D coordinates of two vertices (V_p), leaving the rest of the vertices evolve freely (V_f), $|V_p| + |V_f| = |V|$. The objective function will have the form $E(\mathbf{x}) = \|\mathbf{A} \cdot \mathbf{x} - \mathbf{b}\|^2$, where $\mathbf{x}$ represents the vector of 2D coordinates of the free vertices of the mesh ($\mathbf{x} \in \mathbb{R}^{2|V_f|}$), A is a sparse matrix containing the conformality conditions ($\mathbf{A} \in \mathbb{R}^{2|T| \times 2|V_f|}$). $\mathbf{b} \in \mathbb{R}^{2|T|}$ is the vector that introduces the 2D coordinates of the fixed vertices into the system. Considering that the system is defined in terms of 2D coordinates, other positional constraints can be easily added to it, the result being a trade-off between the conformality of the mapping and the imposed constraints. By adding the rotation equations (2) that impose affine transformations for sister cuts (Fig. 4), the minimization energy will become: $E_{LSGP} = E_{LSCM} + E_{Rot}$, where:

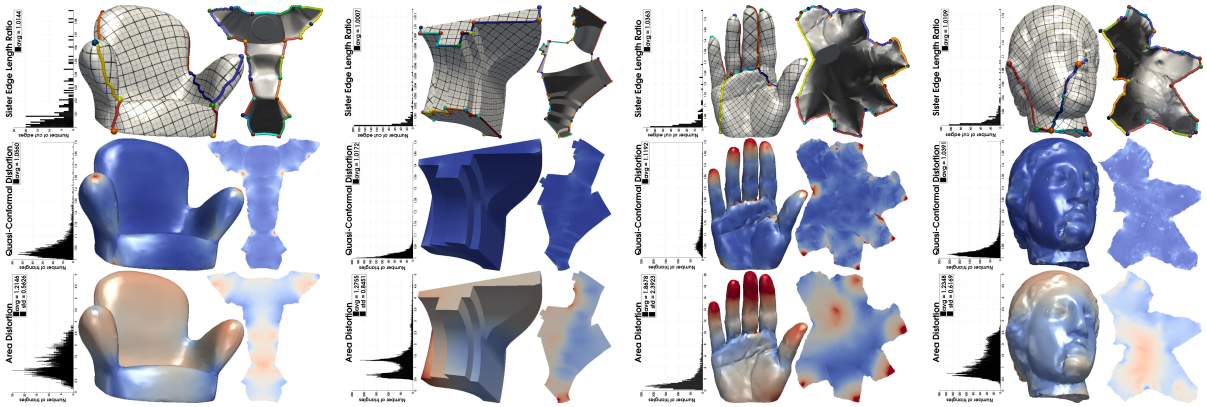
$$E_{Rot} = \sum_{CP_i \subset CP} \left(\sum_{V \subset CP_i} \left\| U_{V_{S_2}} - (T1 \cdot R \cdot T2)\, U_{V_{S_1}} \right\|^2 \right) \tag{6}$$

The number of additional equations will be equal to the number of duplicated vertices V_d from each conic cut CP_i multiplied by two (one equation for each of the two planar coordinates - u and v). Therefore in the linear system, the number of equations will increase ($\mathbf{A} \in \mathbb{R}^{(2|T|+2|V_d|) \times 2|V_f|}$), while the number of variables remains the same.

In the case of straightness conditions (Sec. 5), the employed minimization energy is: $E_{Curvature\_Precription} = E_{LSCM} + E_{Str}$, where:

$$E_{Str} = \sum_{P_i \subset P} \left(\sum_{V_j \subset P_i} \left\| U_{V_j} - U_{C1} \cdot \left(1 - r^j_{CP}\right) - U_{C2} \cdot (r^i_{CP}) \right\|^2 \right) \tag{7}$$

where P represents the entire set of paths resulted after the cutting. Therefore the total number of equations will be $2|T|$ to which we add $2\left(\sum_{P_i \subset P}(|P_i| - 2)\right)$ straightness equations, corresponding to the total number of vertices along paths, except their ending points; which is equal to $(|V_{bdry}| - 2|P|)$, where V_{bdry} represents the total number of boundary vertices after the mesh has been cut. To remove the need for positional constraints of at least two vertices as in LSCM [Lev02], we employ the spectral approach described by Mullen et al. [Mul08].



(a) *Armchair* (b) *Fandisk* (c) *Hand* (d) *Hygeia*

Figure 7: Global parameterization examples obtained with our algorithm. From top to bottom: *(i)* textured surface with cones and cuts (with planar domain inset), *(ii)* quasi-conformal distortion, *(iii)* area distortion. Inset histograms on the left indicate, from top to bottom: sister edge length, quasi-conformal and area distortions (ideal values: 1). Our approach yields global conformal parameterizations with low area distortion.

7 EXPERIMENTS

Experiments were performed with a C++ implementation of our approach (using Eigen, Spectra and Boost libraries), on a laptop with a 2.50GHz i7-4710HQ CPU. Our test data-sets were taken from the AIM@SHAPE repository, [AIM].

7.1 Quality Estimations

We evaluated the quality of our approach with respect to the following quantitative measures. They should ideally be all equal to 1.

1. *Quasi-conformal distortion [Kha06]:* ratio between the largest and smallest eigenvalues of the metric tensor of the parameterization. This indicates a violation of the conformality condition. The color code map depicted in Fig. 7 and Fig. 9 ranges from blue (1) to red (1.5).

2. *Area distortion:* ratio between the normalized area of a triangle in 3D and in 2D (the normalized area refers to the proportion between the area of a triangle and the total area of the mesh). This indicates how much the surface needs to be stretched to be unfolded. The color code map depicted in Fig. 7 and Fig. 9 ranges from blue (0) to red (5).

3. *10^{th} and 90^{th} Percentile of Area distortion:* area distortion values below which (and above which) are located the top 10% triangles that have been scaled down (and scaled up respectively) the most after parameterization.

4. *Sister edge length distortion*: ratio between the planar lengths of a conic cut edge and its sister's. This indicates the violation of the continuity of the global parameterization.

5. *L2Stretch:* measure of distance preservation, computed as in [San01].

Tab. 1 reports a comparison, with respect to these measures, between our technique and Least-Squares Conformal Maps (LSCM) [Lev02], to which conic cuts have been applied in a pre-process (as in Fig. 3(b)), for the sake of a fair comparison. Although slightly higher than those of LSCM, the quasi-conformal distortion measures obtained by our approach are reasonable, Fig. 7: in all our experiments, the worst quasi-conformal distortion is 1.119, Fig. 7(c). For all the remaining criteria - area distortion, sister edge length distortion, L2stretch - our approach outperforms LSCM with conic cuts for all surface examples but one. Although our approach balances conformality for globality, it still produces a quasi-conformal distortion that is on par with Least-Squares Conformal Maps (LSCM), while improving on the area distortion, Fig. 3. The sister edge length distortion is very close to 1.0 on all models, demonstrating the good globality of our parameterization. This is further exemplified in Fig. 3(c), where the area distortion is indeed continuous across the conic cuts (in contrast to LSCM with conic cuts Fig. 3(b)).

7.2 Time Requirement

Tab. 2 presents the running times for the different steps of our approach on the surfaces shown in the paper. This table also presents the total runtime of our approach and the runtime of the LSCM approach [Lev02] with conic cuts, including mesh cutting, as well as the speedup. As illustrated, the most expensive steps of our approach are the linear solvers for the curvature prescription and the final unfolding. Note that even combined, these two steps are still faster than the original LSCM approach. We suspect this performance

Model	$\|T\|$	$\|C\|$	Distortion											
			Quasi Conformal		Areal		10^{th} Percentile Areal		90^{th} Percentile Areal		Sister Edge Length		L2Stretch	
			Our Approach	LSCM [Lev02]	Our Approach	LSCM [Lev02]	Our Approach	LSCM [Lev02]	Our Approach	LSCM [Lev02]	Our Approach	LSCM [Lev02]	Our Approach	LSCM [Lev02]
Octa-flower (Fig. 2)	16K	6	1.016	1.016	**0.965**	1.089	**0.808**	0.661	**1.166**	1.428	**1.002**	1.294	**1.010**	1.063
Planck (Fig. 3)	47K	8	1.026	**1.021**	**1.488**	1.575	0.516	**0.525**	3.370	**3.295**	**1.007**	1.711	1.273	1.273
Fandisk (Fig. 7(b))	13K	25	**1.017**	1.018	**1.276**	3.601	**0.606**	0.536	**2.068**	8.748	**1.001**	2.609	**1.132**	1.926
Armchair (Fig. 7(a))	5K	12	1.056	**1.042**	**1.215**	1.560	**0.633**	0.525	**1.961**	3.156	**1.014**	1.626	**1.099**	1.252
Hand (Fig. 7(c))	5K	11	1.119	**1.085**	**1.868**	4.166	**0.451**	0.350	**4.134**	8.445	**1.036**	1.365	**1.455**	2.010
Hygeia (Fig. 7(d))	16.5K	14	1.039	**1.037**	**1.235**	1.521	**0.618**	0.462	**2.123**	3.013	**1.011**	1.448	**1.111**	1.247

Table 1: Comparison of distortion measures between our approach and Least Squares Conformal Maps [Lev02] with conic cuts. For each criterion (ideal values: 1), the best measure of the two techniques is displayed in bold.

Model	$\|T\|$	$\|C\|$	Runtime [s]							Speedup of our approach vs. LSCM [Lev02]
			Mesh Cutting (Sec. 4.1)	Curvature Prescription (Sec. 5)		Rotationally Constrained Unfolding (Sec. 4.2)		Our approach Total	LSCM [Lev02] Total	
				Setup	Solve	Setup	Solve			
Octa-flower (Fig. 2)	16K	6	0.144	0.051	0.028	0.079	0.014	0.316	2.085	6.6
Planck (Fig. 3)	47K	8	1.096	0.214	0.130	0.477	0.091	2.008	16.760	8.35
Fandisk (Fig. 7(b))	13K	25	0.010	0.053	0.032	0.088	0.029	0.212	1.551	7.32
Armchair (Fig. 7(a))	5K	12	0.017	0.014	0.007	0.019	0.007	0.064	0.324	5.06
Hand (Fig. 7(c))	5K	11	0.017	0.015	0.008	0.017	0.008	0.065	0.235	3.62
Hygeia (Fig. 7(d))	16.5K	14	0.160	0.067	0.033	0.090	0.024	0.374	2.399	6.41

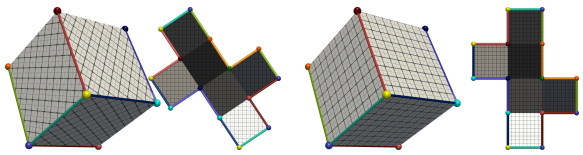
Table 2: Computation times for each step of our approach in seconds.

gain is due to the fact that the original LSCM method uses an indirect method (Conjugate Gradients) to solve the least-squares problem, while we employ the spectral method described by Mullen et al. [Mul08]. The average speedup of our method compared to LSCM, [Lev02] is 6.23.

In comparison to the approach by Myles and Zorin [Myl12], who report a timing of 12.55 seconds for the Fandisk mesh for only the first two steps of their algorithm (cone detection and curvature prescription), our method requires only 0.328 seconds overall.

8 SEAMLESS TEXTURES APPLICATION

As described previously, our approach computes global parameterizations, where area distortion is continuous across conic cuts, Fig. 8(a). For example, this facilitates texture design. Artists want to paint across cuts without noticing distortions. However, for specific texturing tasks such as procedural texturing with periodic patterns, it is additionally beneficial to enforce planar coordinate alignment across the cuts, to guarantee the alignment of the periodic pattern. Such a parameterization is called *seamless* and it is illustrated on a simple cube in Fig. 8(b), where the repeating checker board pattern is indeed well aligned across conic cuts. Seamless parameterizations are also useful for re-meshing as pure quadrangulations can readily be extracted from them. Seamless texturing requires the usage of specific transition functions across conic cuts: translations and rotations by multiples of $\pi/2$. Additionally, cone singularities must be located at integer texture coordinates.

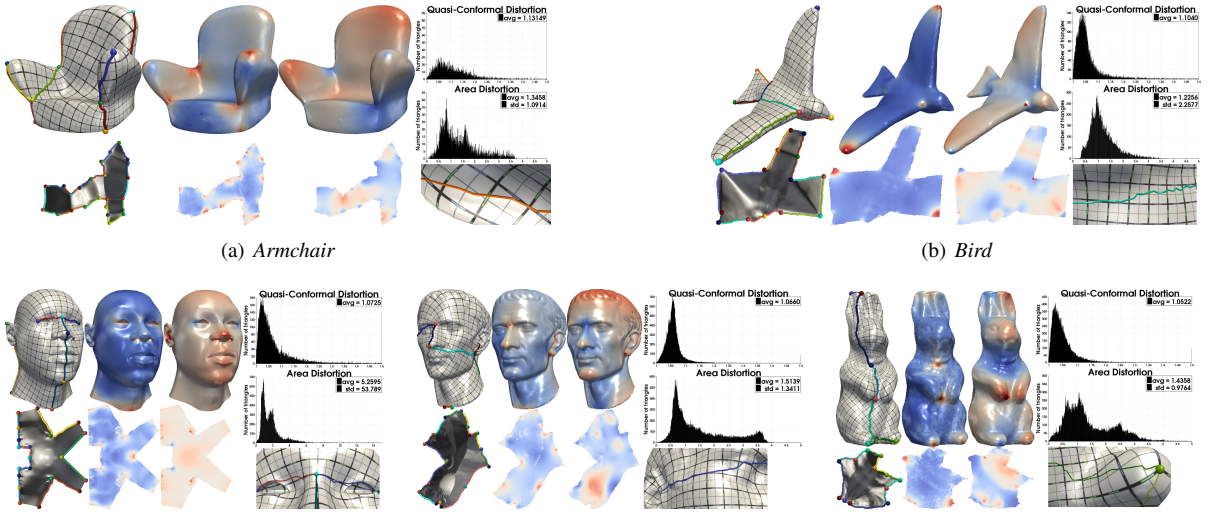


(a) *LSGP* (b) *Seamless LSGP*

Figure 8: Cube unfolding with our approach (a); our approach - seamless (b).

These two constraints can easily be integrated in our approach.

Given input cones as well as initial curvature prescriptions computed automatically by our approach, we start by rounding the curvature prescriptions to the nearest multiples of $\pi/2$ while respecting the Gauss-Bonnet theorem, similarly to Springborn et al. [Spr08]. In particular if the sum of prescribed angles is different from the allowed sum, we decrease them in descending order of their rounding error. The resulting angles are then prescribed in the reminder of the proposed algorithm. Next, we snap each cone to the nearest integer location. The resulting integer locations are then added as hard constraints to the linear system described in Sec. 6 and the rest of the algorithm is executed as is. In the extreme case where several cones are quantized to the same (u,v) coordinates, we relax the seamless constraints by not pinning such cones, but rather adding them to the system as soft constraints. This case appears rarely in practice and it is either caused by the proximity of the cones in 3D, the high cone number or the low resolution of the texture space.



(a) *Armchair* (b) *Bird*

(c) *Head* (d) *Julius* (e) *Rabbit*

Figure 9: Seamless global parameterization obtained with our algorithm. From left to right: *(i)* textured surface with cones and cuts (with planar domain inset), *(ii)* quasi-conformal distortion, *(iii)* area distortion, *(iv)* quasi-conformal and area distortion histograms (ideal values: 1). Our approach yields global conformal parameterizations with low area distortion and seamless texture transitions across conic cuts (inset zooms).

Fig. 2(d), Fig. 1(a), Fig. 3(d), Fig. 8(b) and Fig. 9 provide examples of seamless global parameterizations obtained with our approach. The initial curvature prescription, before rounding, has been automatically evaluated by our method for all surfaces. As showcased in these examples, based on the integration of seamless constraints, our approach provides rapidly seamless global parameterizations with low area distortion.

9 CONCLUSION

We have presented a fast and efficient method for the global parameterization of triangular surfaces.

For modeling the transition functions between pairs of sister conic cuts, we introduced linear equations which account for translations and rotations with given cone angles. Also we have provided an automatic method to compute such angles. Extensive experimental results demonstrate the time efficiency of our algorithm which performs better than standard, non-global parameterization algorithms [Lev02]. The average speedup of our method compared to LSCM is 6.23. The quality of our parameterizations has been illustrated by examining accepted distortion measures. We demonstrated the interest of the computational speed of our approach in the seamless texturing application, which requires slight modifications to our algorithm.

In future work, we want to extend the seamless texturing application. In particular, the proposed method for detecting integer positions for the cones is not guaranteed to find solutions for the entire set of cones, the quantization can be partial, but we show a number of examples where the application provides satisfactory results. A more robust but still efficient quantization of the cones remains an open problem that we will address in the future. Although our procedure for the resolution of conic cut intersection in the planar domain has worked successfully in our experiments, we would like to further investigate theoretical guarantees regarding the bijective property of the maps computed by our approach. Also, since our method handles only disk or sphere topology, another future work direction lies in the extension of our algorithm to surfaces of non trivial topology, by applying loop computation algorithms [Dey08, Dey13].

10 REFERENCES

[Aig15] N. Aigerman and Y. Lipman. Orbifold tutte embeddings. *ACM Trans. Graph.*, 34(6):190:1–190:12, 2015.

[AIM] Digital shape workbench. AIM@SHAPE shape repository. `http://visionair.ge.imati.cnr.it/ontologies/shapes/`

[Ben08] M. Ben-Chen, C. Gotsman, and G. Bunin. Conformal flattening by curvature prescription and metric scaling. *Comput. Graph. Forum*, 27(2):449–458, 2008.

[Bom13] D. Bommes, M. Campen, H.-C. Ebke, P. Alliez, and L. Kobbelt. Integer-grid maps for reliable quad meshing. *ACM Trans. Graph.*, 32(4):98:1–98:12, 2013.

[Bom12] D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, and D. Zorin. State of the art in quad meshing. In *Eurographics STARS*, pp. 159–182.

[Bom09] D. Bommes, H. Zimmer, and L. Kobbelt. Mixed-integer quadrangulation. *ACM Trans. Graph.*, 28(3):77:1–77:10, 2009.

[Cam15] M. Campen, D. Bommes, and L. Kobbelt. Quantized global parametrization. *ACM Trans. Graph.*, 34(6):192:1–192:12, 2015.

[Des02] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. *Comput. Graph. Forum*, 21(3), 2002.

[Dey13] T. K. Dey, F. Fan, and Y. Wang. An efficient computation of handle and tunnel loops via reeb graphs. *ACM Trans. Graph.*, 32(4):1–10, 2013.

[Dey08] T. K. Dey, K. Li, J. Sun, and D. Cohen-Steiner. Computing geometry-aware handle and tunnel loops in 3d models. *ACM Trans. Graph.*, 27(3):45:1–45:9, 2008.

[Ebk13] H. C. Ebke, D. Bommes, M. Campen, and L. Kobbelt. QEx:Robust Quad Mesh Extraction. *ACM Trans. Graph.*, 32(6), 2013. `http://www.rwth-graphics.de/software/libQEx`

[Flo05] M. Floater and K. Hormann. Surface parameterization:a tutorial and survey. In *Adv. Multires. Geom. Mod.* Springer, 2005.

[Jin08] M. Jin, J. Kim, and X. D. Gu. *Discrete Surface Ricci Flow: Theory and Applications*, pp. 209–232. Springer, 2007.

[Kal07] F. Kalberer, M. Nieser, and K. Polthier. Quadcover – surface paramterization using branched coverings. *Comput. Graph. Forum*, 26(3):375–384, 2007.

[Kha06] L. Kharevych, B. Springborn, and P. Schröder. Discrete conformal mappings via circle patterns. *ACM Trans. Graph.*, 2006.

[Kha05] L. Kharevych, B. Springborn, and P. Schröder. Cone singularities to the rescue: Mitigating area distortion in discrete conformal maps. In *Symp. on Geom. Processing*, ACM SIGGRAPH/Eurographics , 2005.

[Kno13] F. Knöppel, K. Crane, U. Pinkall, and P. Schröder. Globally optimal direction fields. *ACM Trans. Graph.*, 32(4):1–10, 2013.

[Lev02] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph.*, 21(3):362–371, 2002.

[Liu08] L. Liu, L. Zhang, Y. Xu, C. Gotsman, and S. Gortler. A local/global approach to mesh parameterization. In *Proc. of the Symp. on Geom. Processing*, SGP '08, pp. 1495–1504, 2008.

[Mul08] P. Mullen, Y. Tong, P. Alliez, and M. Desbrun. Spectral conformal parameterization. In *Proc. of the Symp. on Geom. Processing*, SGP '08, pp. 1487–1494, 2008.

[Myl14] A. Myles, N. Pietroni, and D. Zorin. Robust field-aligned global parametrization. *ACM Trans. Graph.*, 33(4):1–14, 2014.

[Myl13] A. Myles and D. Zorin. Controlled-distortion constrained global parametrization. *ACM Trans. Graph.*, 32(4):1–14, 2013.

[Myl12] A. Myles and D. Zorin. Global parametrization by incremental flattening. *ACM Trans. Graph.*, 31(4):109:1–109:11, 2012.

[Pan12] D. Panozzo, Y. Lipman, E. Puppo, and D. Zorin. Fields on symmetric surfaces. *ACM Trans. Graph.*, 31(4):1–12, 2012.

[Ray08] N. Ray, B. Vallet, W. C. Li, and B. Lévy. N-symmetry direction field design. *ACM Trans. Graph.*, 27(2):10:1–10:13, 2008.

[Ray06] N. Ray, W. C. Li, B. Lévy, A. Sheffer, and P. Alliez. Periodic global parameterization. *ACM Trans. Graph.*, 25(4), 2006.

[Ray03] N. Ray and B. Lévy. Hierarchical least squares conformal map. In *Pacific Graphics*, 2003.

[San01] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe. Texture mapping progressive meshes. ACM SIGGRAPH, 2001.

[She06] A. Sheffer, E. Praun, and K. Rose. Mesh parameterization methods and their applications. *Found. Trends. Comput. Graph. Vis.*, 2(2):105–171, 2006.

[She05] A. Sheffer, B. Lévy, M. Mogilnitsky, and A. Bogomyakov. Abf++: Fast and robust angle based flattening. *ACM Trans. Graph.*, 24(2):311–330, 2005.

[She00] A. Sheffer and E. De Sturler. Surface parameterization for meshing by triangulation flattening. In *Proc. of IMR*, 2000.

[Spr08] B. Springborn, P. Schröder, and U. Pinkall. Conformal equivalence of triangle meshes. *ACM Trans. Graph.*, 27(3), 2008.

[Ton06] Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. Designing quadrangulations with discrete harmonic forms. In *Symp. on Geom. Processing*, pp. 201–210, 2006.

[Yan09] Y. L. Yang, R. Guo, F. Luo, S. M. Hu, and X. Gu. Generalized discrete ricci flow. *Comput. Graph. Forum*, 28(7), 2009.

[Zay07] R. Zayer, B. Lévy, and H.-P. Seidel. Linear angle based parameterization. In *Symp. on Geom. Proc.*, pp. 135–141, 2007.

Glyphs for Space-Time Jacobians of Time-Dependent Vector Fields

Tim Gerrits Christian Rössl Holger Theisel
Visual Computing Group, University of Magdeburg
{gerrits|roessl|theisel}@isg.cs.uni-magdeburg.de

ABSTRACT

Glyphs have proven to be a powerful visualization technique for general tensor fields modeling physical phenomena such as diffusion or the derivative of flow fields. Most glyph constructions, however, do not provide a way of considering the temporal derivative, which is generally nonzero in non-stationary vector fields. This derivative offers a deeper understanding of features in time-dependent vector fields. We introduce an extension to 2D and 3D tensor glyph design that additionally encodes the temporal information of velocities, and thus makes it possible to represent time-dependent Jacobians. At the same time, a certain set of requirements for general tensor glyphs is fulfilled, such that the new method provides a visualization of the steadiness or unsteadiness of a vector field at a given instance of time.

Keywords

Tensor Glyphs, Glyph Design, Flow Visualization

1 INTRODUCTION

Glyphs have gained popularity as a tool for investigating second-order tensors and their properties. They offer a convenient way to represent some of the underlying physical meaning encoded in tensors such as diffusion or stress tensors at a given location in the data. The Jacobian matrix $\mathbf{J}$ of velocity fields is a second-order tensor, which appears in flow visualization and describes the local behavior of the flow at a given location, possibly in space-time. Unlike diffusion tensors, the second-order tensor Jacobians appear as general square matrices, including in particular non-symmetric matrices. This is generally a 2×2 matrix in 2D space, or 3×3 in 3D that consists of the spatial partial derivatives.

Finding appropriate visualization techniques to represent these matrices has proven to be a challenging task. Seltzer and Kindlmann [11] proposed the first glyphs for 2D tensor data that are able to represent any second-order tensor using the information encoded by eigenvalues and eigenvectors. Recently, Gerrits et al. [3] proposed a construction of glyphs for 2D tensors, which was then extended to visualize general 3D tensors as well. However, considering the special case of time-dependent Jacobian matrices, both approaches only deal with spatial derivatives and neglect temporal information that might be available.

In this paper, we solve the following problem: given an n-dimensional ($n = 2,3$) time-dependent vector field $\mathbf{v}(\mathbf{x},t)$, we construct an n-dimensional glyph that encodes the *space-time* Jacobian matrix of $\mathbf{v}$, i.e., all first order derivatives, both spatial and temporal, of $\mathbf{v}$. This means that we have to find a glyph representation for a $(n+1)\times(n+1)$ Jacobian tensor. While this is straightforward for $n = 2$ (ending up in visualizing a 3×3 matrix), it is challenging for $n = 3$ because this requires the 3D visualization of a 4×4 space-time Jacobian tensor. We show that this tensor, which is not a general 4D second-order tensor, has some properties that allow a 3D glyph visualization that seamlessly extends existing 3D tensor glyphs.

After analyzing related work in section 2, we present an extension of an existing second-order tensor glyph construction in section 3 that includes the temporal derivative and therefore offers, to the best of our knowledge, the first glyph for time-dependent 2D and 3D Jacobians. This extension is constructed to in no way impair the glyph's capability to encode the spatial derivatives. It becomes invisible, when the temporal derivative vanishes. In this case, the resulting glyphs become identical to time-independent tensor glyphs.

The results shown in section 4 present our final glyph designs for 2D and 3D time-dependent Jacobians. Applying them on sampled locations demonstrates how the resulting glyphs additionally encode the temporal derivative of given time-dependent vector fields.

2 RELATED WORK AND BACKGROUND

2.1 Tensor Glyphs Visualization

Using visualization of flow fields to gain further insight of the underlying behavior has produced a huge variety of techniques that can coarsely be classified in different groups: topology-based techniques [9], dense flow visualization [7], geometric flow visualization [8] and glyph-based approaches. Glyphs are a convenient visualization technique, as they are often tailored to fit a specific application and offer a seemingly unlimited design space, see Boro et al. [1]. Research trying to develop glyphs for second-order tensors has mostly been limited to symmetric positive definite matrices. The pioneering work by Schultz and Kindlmann [10] uses superquadrics to create glyphs, where shape and orientation are defined by the eigenvectors of general symmetric tensors, including indefinite matrices. This is convenient, as eigenvectors of symmetric matrices are always orthogonal and therefore easily usable to define appropriate shapes. A general discussion on different approaches has been given by Kratz et al. [6]. Seltzer and Kindlmann [11] recently presented glyphs for general – symmetric and asymmetric – second-order 2D tensors, extending the superquadrics. Gerrits et al. [3] propose a different approach that provides glyphs also for general second-order tensors in 3D. Both works offer an in-depth discussion of tensor properties and design principles leading to a set of requirements for a suitable glyph which will be covered in more detail in the next section.

As opposed to steady flows, time-dependent flow fields are only sparsely covered by glyph-based approaches. Often pathlines of a finite set of seed points are used to visualize flow in this case. Especially in topology-based visualization techniques, several works have been proposed, where the path of features over time is visualized as presented by Uffinger and Sadlo [13]. And even though there exist several approaches that try to extract only meaningful selections to give further insight [14, 4], there is still often visual clutter, overlapping elements or missing information by only rendering selected features. Hlawatsch et al. [5] downscale pathlines to represent them as so called pathline glyphs thus combining both techniques to address this problem.

2.2 Glyph Construction

Schultz and Kindlmann [10] present a set of construction principles to build glyphs for symmetric tensors, which includes *preservation of symmetry*, *continuity*, *disambiguity*, *invariance under scaling* as well as *eigenplane projection*. The last requirement, however, is not well defined for asymmetric tensors, which is why Gerrits et al. [3] present a similar set of properties, but the latter is replaced by the demand for *direct encoding of real eigenvalues and eigenvectors*. As these properties also influence the choices made in this paper, they are listed and explained in short:

(a) *Invariance under isometric domain transformation*: any isometric transformation of the domain should result in the same isometric transformation of the glyph's shape.

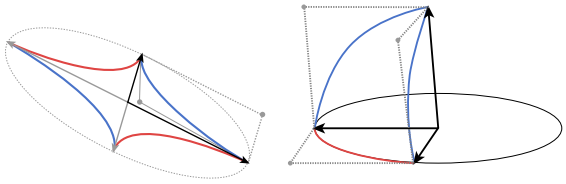
(b) *Scaling invariance*: a uniform scaling of the tensor has to result in the same scaling of the glyph for any real positive scaling value.

(c) *Direct encoding of real eigenvalues and eigenvectors*: all real eigenvalues and eigenvectors of the tensor should be directly visible within the shape of the glyph.

(d) *Uniqueness*: a tensor should be represented by a unique corresponding glyph and vice-versa, such that for any two dissimilar tensors no similar glyph is produced and no dissimilar glyphs are produced by the same tensor.

(e) *Continuity*: any continuous change of the tensor should result in a continuous change of the glyph, preventing abrupt alterations of the appearance for small changes.

A glyph that satisfies all of these properties cannot be encoded by shape alone, but also needs at least one additional channel such as color.



(a) (b)

Figure 1: Basic glyph construction. (a) In 2D, each pair of scaled eigenvectors (↗) is interpolated by four rational quadratic Bézier curves (╱ and ╱). (b) The 3D case relies on the 2D construction in a base plane (╱) and triangular surface patches interpolating the 2D curves and the third scaled eigenvector.

For a 2D tensor $\mathbf{J} \in \mathbb{R}^{2\times2}$ with real eigenvalues, the eigenvectors scaled by eigenvalues define an interpolating ellipse. Note that the eigenvectors are not necessarily orthogonal, which is the case only for symmetric tensors. The geometric construction of the glyphs is based upon modifying this ellipse, such that the directions of the scaled eigenvectors are encoded by the shape. In [3], this ellipse is parametrized by four rational quadratic curves in Bernstein-Bézier form ([2]), and the center control points and rational weights are modified to express properties of the glyph. Figure 1a shows the construction for a "saddle" configuration with two

real eigenvalues with opposite sign, which results in a concave shape. The sharp corners at the break points between the rational pieces encode the direction of eigenvectors and magnitude of eigenvalues. The ellipse is also well-defined in case of non-real, i.e., a pair of complex conjugate eigenvalues: then the right singular vectors replace the eigenvectors in the construction. The transition between both cases is continuous.

Color is used to indicate the sign of real eigenvalues and rotation for complex eigenvalues. With shape and color these glyphs are capable of uniquely representing every possible 2D tensor such as the Jacobian of a steady vector field. Moreover, they provide an intuitive interpretation:

Convex shapes indicate that the eigenvalues share the same sign, whereas *concave shapes* imply that the eigenvalues have different signs. The color additionally illustrates the sign of the corresponding eigenvalue. Moreover, discontinuities of the boundary curve, i.e., "sharp corners", indicate direction and scale of eigenvectors. Figure 2 shows examples.

Figure 2: The glyph's shape indicates the relation of both eigenvalue signs. A red glyph has two positive, the blue one two negative eigenvalues. They are therefore convex shapes. If the eigenvalues have opposite eigenvalues, the shape is concave.

An *ellipse* without discontinuities indicates that there are no unique eigenvectors as the eigenvalues are either identical – the shape is a circle – and/or complex. In the latter case, the rotation is encoded by different colors.

Figure 3: Ellipses indicate identical and/or complex eigenvalues. A perfect sphere indicates two identical eigenvalues, whereas ellipses represent rotational behavior. Colors close to yellow indicate counterclockwise, those close to green clockwise rotation.

In 3D, an additional eigenvector and eigenvalue needs to be visually encoded by the glyph. The construction is partially based on the 2D configuration: eventually, two eigenvectors (or two left singular vectors in case of a complex conjugate pair of eigenvalues) span a supporting base plane, in which the 2D construction is applied. The 2D curves in the plane are used together with the remaining real eigenvector to setup a shape made of surface patches. Figure 1b illustrates the construction of one patch, and figure 4 shows examples of 3D glyphs with a similar color coding as for the 2D case. This review is simplified, the different cases depend on the eigenvalues. For an in-depth view on the construction, please refer to [3].

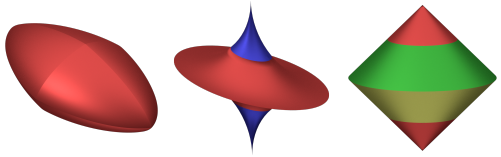


Figure 4: Glyphs representing different 3D Jacobians. From left to right: All eigenvalues are positive; The two positive eigenvalues span the base plane and the third, negative one makes for the concave shape; The base plane indicates rotational behavior in the corresponding plane and additional outflow.

3 EXTENSION FOR TIME-DEPENDENT TENSOR GLYPHS

The glyphs shown in the previous section can visualize any given 2D or 3D Jacobian as long as the feature is steady. We therefore use them as a construction foundation to build upon. They need to be altered or extended in some way, such that they are able to represent the additional information encoded in time-dependent Jacobians. To find a suitable extension, we need to analyze the differences between the steady and unsteady case and discuss, how a suitable mapping of the additional data to the same dimension as the glyphs we build upon can be found. First, we do this for Jacobians of 2D unsteady vector fields and present a simple addition to the given glyph, following a set of requirements and later extend the idea to the 3D case.

3.1 Time Dependent 2D Tensor Glyphs

A steady 2D flow is given by

$$\mathbf{v}(x,y) = \begin{pmatrix} u(x,y) \\ v(x,y) \end{pmatrix},$$

where the Jacobian matrix $\mathbf{J}$ is defined as

$$\mathbf{J}(x,y) = \begin{pmatrix} u_x & u_y \\ v_x & v_y \end{pmatrix}$$

This is the spatial gradient of the vector field and hence the *spatial Jacobian*. Using eigendecomposition, we obtain the eigenvalues λ_1, λ_2 and the corresponding eigenvectors $\mathbf{e}_1, \mathbf{e}_2$. An unsteady flow, however, has time as an additional dimension. We define

$$\bar{\mathbf{v}}(x,y,t) = \begin{pmatrix} u(x,y,t) \\ v(x,y,t) \\ 1 \end{pmatrix}$$

to be a time-dependent 2D vector field and the corresponding *space-time Jacobian* (see, e.g., [15]) as

$$\bar{\mathbf{J}}(x,y,t) = \begin{pmatrix} u_x & u_y & u_t \\ v_x & v_y & v_t \\ 0 & 0 & 0 \end{pmatrix}$$

with eigenvalues λ_1, $\lambda_2, 0$. The associated eigenvectors are

$$\begin{pmatrix} \mathbf{e}_1 \\ 0 \end{pmatrix}, \begin{pmatrix} \mathbf{e}_2 \\ 0 \end{pmatrix}, \quad \bar{\mathbf{f}} \text{ where } \bar{\mathbf{f}} =: \begin{pmatrix} a \\ b \\ c \end{pmatrix}.$$

This Jacobian must not be mistaken with a general 3×3 matrix. Due to the fact, that the last row of $\bar{\mathbf{J}}$ is entirely made up of zeros, two of these eigenvectors are simply the eigenvectors $\mathbf{e}_1$ and $\mathbf{e}_2$ of $\mathbf{J}$ with an additional zero as their component in the new dimension. The additional eigenvector $\bar{\mathbf{f}}$ with its components $a, b, c \in \mathbb{R}$ is associated with the zero eigenvalue and fully encodes the temporal derivative, included in $\bar{\mathbf{J}}$. We can therefore use $\mathbf{e}_1$ and $\mathbf{e}_2$ to build the corresponding 2D glyph, which we call spatial glyph, and use only $\bar{\mathbf{f}}$ to be somehow added to it. As we want our new glyph to be of the same dimension as the spatial glyph, we require a projection of $\bar{\mathbf{f}} \in \mathbb{R}^3$ to a vector $\mathbf{g} \in \mathbb{R}^2$ on the visualization plane. To define an appropriate and unique projection, we demand

1. Given two eigenvectors $\bar{\mathbf{f}}_1$, $\bar{\mathbf{f}}_2$ corresponding to the temporal derivative and the projected 2D vectors $\mathbf{g}_1$, $\mathbf{g}_2$, if $\bar{\mathbf{f}}_1$ and $\bar{\mathbf{f}}_2$ are parallel, $\mathbf{g}_1$ and $\mathbf{g}_2$ have to be identical.

$$\bar{\mathbf{f}}_1 \parallel \bar{\mathbf{f}}_2 \quad \Rightarrow \quad \mathbf{g}_1 = \mathbf{g}_2 .$$

2. If $\bar{\mathbf{f}}_1$ and $\bar{\mathbf{f}}_2$ are not parallel, $\mathbf{g}_1$ and $\mathbf{g}_1$ must never be identical

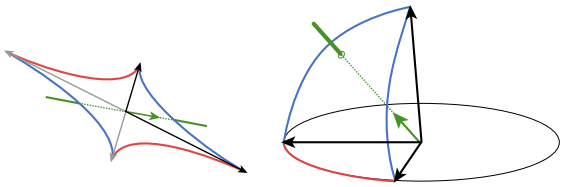
$$\bar{\mathbf{f}}_1 \nparallel \bar{\mathbf{f}}_2 \quad \Rightarrow \quad \mathbf{g}_1 \neq \mathbf{g}_2 .$$

3. When the field is stationary, $\mathbf{g}$ should not be visible. In this case, the resulting glyph is identical to the glyph based on the stationary Jacobian $G(\mathbf{J}) = G(\bar{\mathbf{J}})$. Therefore, the corresponding vector $\mathbf{g}$ should be the null vector. Additionally, the transition from unstable to stable should result in a smooth transition to the null vector.

$$\bar{\mathbf{f}} \to \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad \Rightarrow \quad \mathbf{g} \to \begin{pmatrix} 0 \\ 0 \end{pmatrix} .$$

We propose the following projection that satisfies the above requirements:

$$\mathbf{g} = \frac{1}{||\bar{\mathbf{f}}||} \begin{pmatrix} a \\ b \end{pmatrix},$$



(a) (b)

Figure 5: Adding sticks to the base glyphs to allow time-dependent glyphs to be represented. The eigenvector $\bar{\mathbf{f}}$ corresponding to the time derivation is projected onto the vector $\mathbf{g}$ (↗). A line cast from the center of the glyph in forward and backward direction of $\mathbf{g}$ intersects the boundary of the glyph exactly twice, unless $\mathbf{g}$ is the zero vector. Sticks (╱) representing $\mathbf{g}$ and -$\mathbf{g}$ are then added at those locations. (a) Construction of the 2D time-dependent Jacobian glyph. (b) Construction of one patch and one stick representing the eigenvalue $\bar{\mathbf{f}}$ of an 3D time-dependent Jacobian glyph.

where a and b are the first two components of $\bar{\mathbf{f}}$.

This vector is then visualized by adding two identical sticks to the glyph, one representing $\mathbf{g}$, the other $-\mathbf{g}$ and both given a length of $s||\mathbf{g}||$, where $s > 0 \in \mathbb{R}$ can be used as a constant scaling factor. Rendering both orientations of $\mathbf{g}$ is due to the fact, that $\bar{\mathbf{f}}$ is an eigenvector of $\bar{\mathbf{J}}$, and therefore satisfies the same symmetry properties. To reduce visual clutter, we move these sticks along the lines, given by their directions to the locations where the line intersects the boundary of the underlying spatial glyph's shape. Figure 5a illustrates this construction.

3.2 Time-Dependent 3D Tensor Glyphs

Finding new glyphs representing 3D time-dependent Jacobians is analogous to the 2D case. The additional temporal information encoded by the Jacobian $\bar{\mathbf{J}} \in \mathbb{R}^{4\times4}$ is given by the additional eigenvector $\bar{\mathbf{f}} \in \mathbb{R}^4$, where $\bar{\mathbf{f}} = (a\ b\ c\ d)^{\mathrm{T}}$.

We propose projecting $\bar{\mathbf{f}}$ onto the 3D vector $\mathbf{g} \in \mathbb{R}^3$ by using

$$\mathbf{g} = \frac{1}{||\bar{\mathbf{f}}||} \begin{pmatrix} a \\ b \\ c \end{pmatrix},$$

and visualizing it by adding tubes to the spatial glyph, created by using eigenvalues and eigenvectors of $\mathbf{J}$. These tubes are then moved along their vector directions as well, until they reach the points, where their corresponding line would intersect the glyph patch. In that way, they are always visible and not rendered within the glyph, unless the temporal derivative is zero, in which case the new vector becomes the zero vector as demanded.

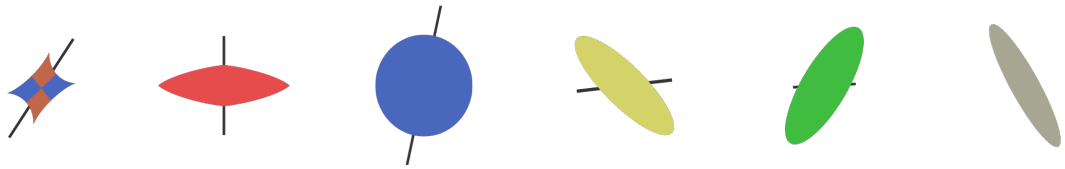


Figure 6: Glyphs representing different 2D Jacobians. The underlying features are less temporally stable to the left and more stable to the right. The stick has vanished in the last glyph, which shows that this feature is completely stable.

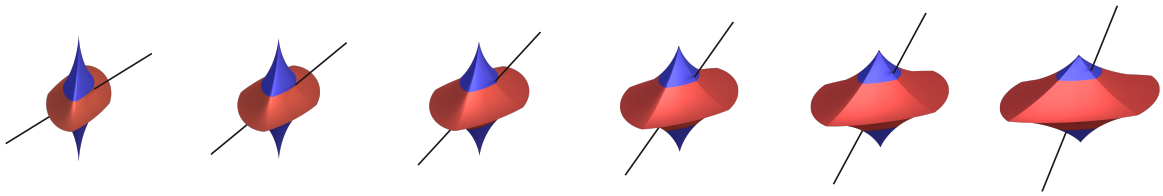


Figure 7: 3D Glyphs representing the same location in an unsteady flow field over time. The glyph as well as the stick representing the temporal derivative change smoothly over time.

Because both new constructions, 2D and 3D alike, follow the presented set of rules, they are suitable for creating unique tensor glyphs for any given 2D or 3D Jacobian, unsteady or steady, and also follows all of the glyph design requirements that were discussed earlier.

4 RESULTS

First, we visualize different 2D time-dependent Jacobians. Figure 6 shows a selection of 2D glyphs for randomly chosen time-dependent 2D Jacobians with decreasing temporal derivative from left to right. These include glyphs based upon real-valued as well as complex-valued eigenvalues and eigenvectors. The additional sticks are always moved to the boundary of the spatial glyph, for any given shape.

In figures 9 and 10, our construction is applied to build glyphs representing the Jacobians at sampled locations of one time slice of a 2D unsteady flow behind a cylinder. This is a sufficiently complex choice as a whole variety of different features is present as can be seen by the variety of different spatial glyphs. As the time proceeds, alternating vortices, as illustrated by the glyphs using yellow and green colors, are created and transported to the right. Therefore, Jacobians at several locations comprise strong temporal derivatives, indicated by the additional sticks being clearly visible. Locations where the derivative vanishes are analogously indicated by small or even no sticks. While figure 9 shows the glyphs superimposed to an additional line integral convolution (LIC) texture of the underlying flow field, figure 10 displays the same glyphs in front of a different LIC texture which in this case represents the *feature flow field* [12] at the selected time. The projected additional eigenvectors are therefore tangent to this field at the given location. Two closeups for each field show zoomed-in areas of interest inside those fields.

To further highlight the sticks, the same domain is rendered without any supporting background LIC texture in figure 11.

Figure 7 demonstrates the new 3D glyphs, as it shows sampled time steps of the development of a 3D Jacobian at the same location evolved over time. The underlying changing Jacobian is computed by linear interpolation of two vector field time slices. The spatial glyph changes independent of the time derivative, whereas the added tubes change direction due to the projected vector, but change location due to change of glyph shape, as seen in figure 8.

In figure 12, the glyphs are used to visualize regularly sampled locations in the 3D unsteady Jacobian field of an analytical flow with one moving center in the middle of the field. The whole flow is steadily moved to the right over time. To illustrate the underlying flow field, a set of illuminated streamlines is added. Here, too, the new glyphs show a variety of different underlying Jacobians, including constructions based upon tensors with complex and real-valued eigenvalues.

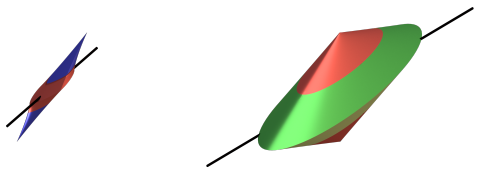


Figure 8: Sticks visualizing the temporal derivative are always moved along their directions to the boundary of the glyph, so they are always visible, no matter whether the spatial glyph is small (left) or large (right).

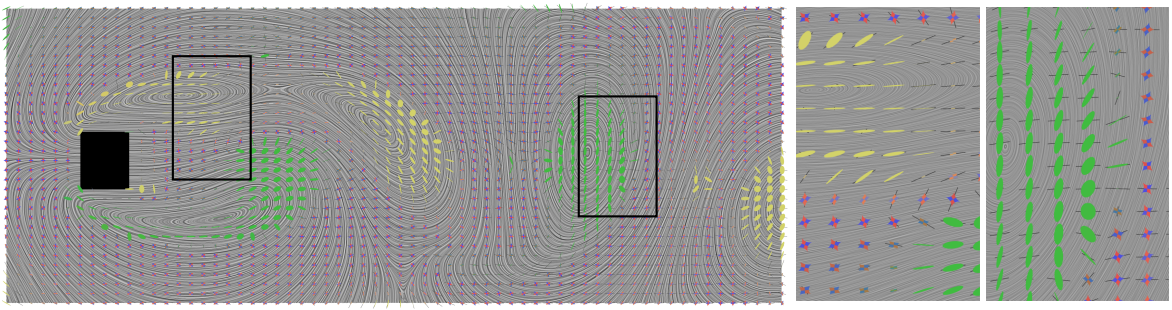


Figure 9: Glyphs visualizing the Jacobian matrix of the flow around a square cylinder with two closeups (rectangles). The underlying LIC image visualizes the fluid flow.

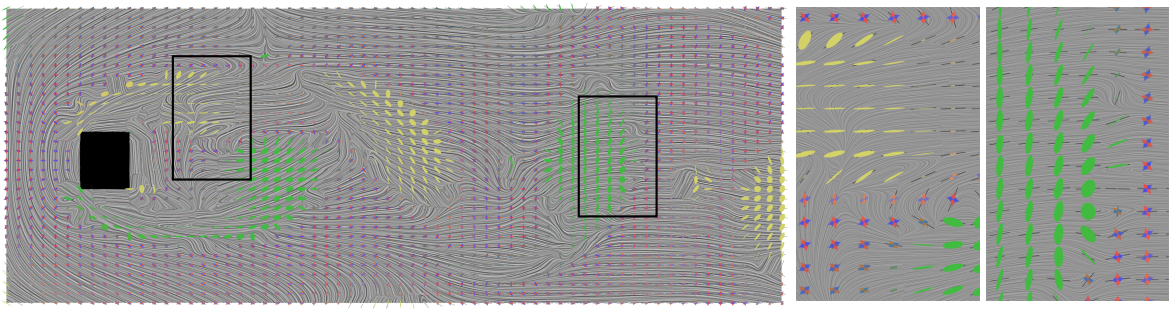


Figure 10: Glyphs visualizing the Jacobian matrix of the flow around a square cylinder with two closeups (rectangles). The underlying LIC image visualizes the feature flow.

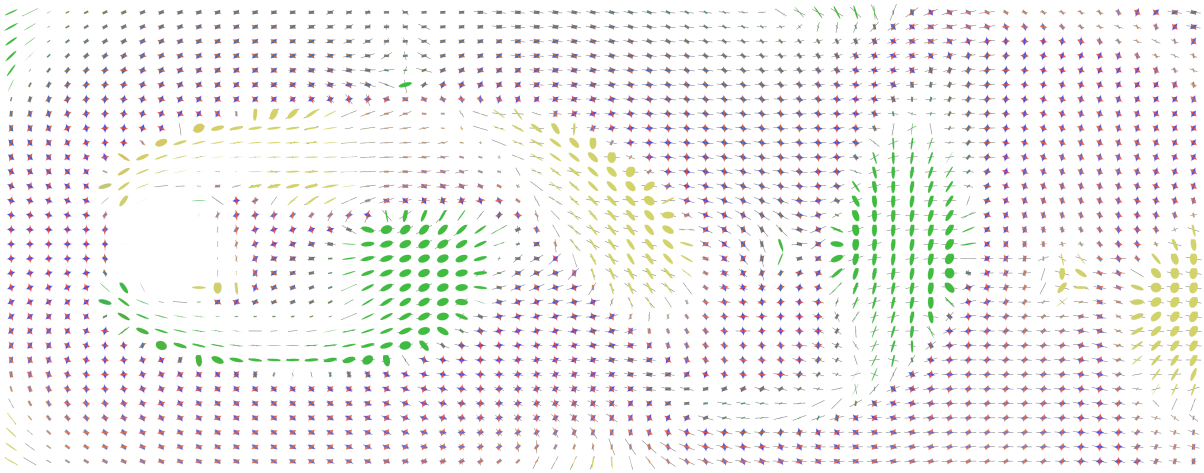


Figure 11: Glyphs visualizing the Jacobian matrix of the flow around a square cylinder without additional supporting LIC textures.

5 DISCUSSION

Looking at figure 9, our newly designed 2D glyphs allow to see the same structures of the underlying flow field as the steady or spatial 2D glyphs before. By finding a mapping onto the same visualization plane and moving it on the shape boundaries, encoding the additional temporal information has not changed the spatial glyph. Therefore, rotational sections as well as laminar flows can still be easily determined in the given example flow. The same statement holds for the 3D case as displayed in figure 12. Even though, the addition of a stick or tube respectively, is only a small extension to the known glyphs, it is one, that does not impair the expressiveness of the spatial glyph and offers a visualization technique for any 2D or 3D time-dependent Jacobian, symmetric or asymmetric.

As the sticks or tubes added to the glyphs are rendered in both directions of the vector, they not only follow the mathematical nature of eigenvector symmetry, they are also always visible, even in the 3D case, regardless of the point of view as long as the underlying Jacobian is unsteady. An appropriate scaling factor or a change of

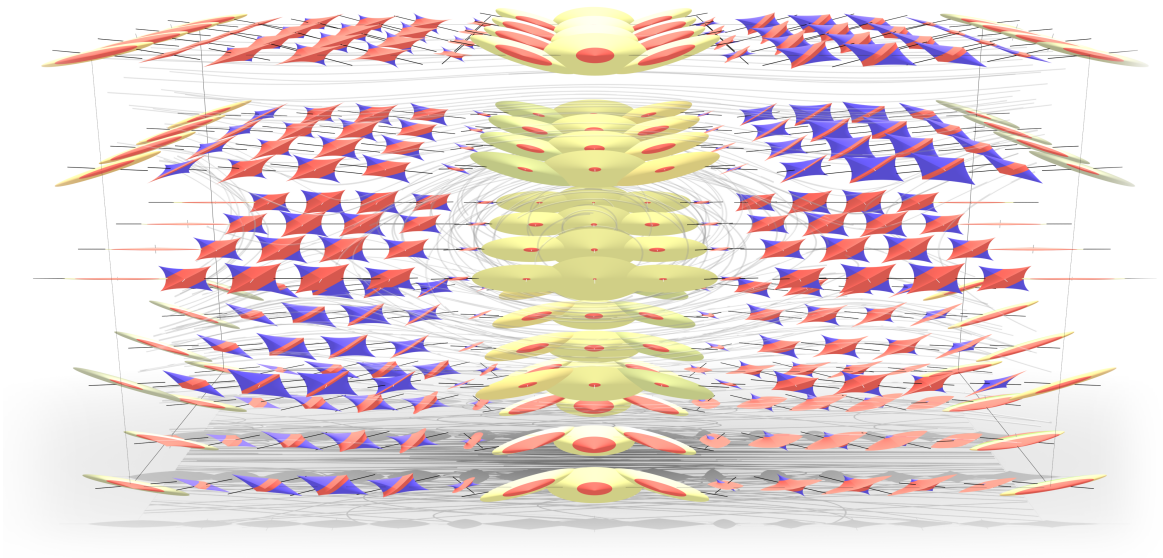


Figure 12: 3D time-dependent flow with a moving vortex in the center. All features are moving to the right over time. The newly constructed glyphs are rendered at sampled locations at one time slice. Illuminated streamlines illustrate the underlying flow.

thickness can be chosen to further emphasize this addition if necessary.

This work focuses on finding a construction of glyphs for general 2D and 3D time-dependent Jacobians while meeting a set of specific design requirements. The transition between the different Jacobian glyphs is smooth, including a change of vector direction or vector length, which is displayed in figure 7, where a time series of the glyphs at the same location over time is shown. This allows our new construction method to seamlessly build upon the building requirements for the initial glyphs, and simply extend them.

When rendered on top of the feature flow field LIC texture, as seen in figure 10, the glyph's sticks are always tangent to it at the sampling location. This field allows tracking critical points over time (see, e.g., [12]) and therefore offers an insight of the progression of flow feature. We can predict glyphs with longer sticks to be moving or changing over time, while shorter or no sticks indicate that a feature is quite stable. The shown flow around the cylinder has vortices going along one axis to the right, which is also indicated by the sticks of the glyphs in those areas pointing in this direction. We can remove all supporting LIC textures as in figure 11 and still understand the flow itself, visualized by the spatial glyphs at the same time as the feature flow encoded by the additional sticks. Inquiring the analytic flow shown in figure 12, all the glyphs indicate this behavior by having tubes added to them, similar in length and direction, as the whole underlying flow is moving horizontally along one axis over time.

6 LIMITATION AND FUTURE WORK

Even though these extensions for the general second-order tensor glyphs can be applied to any temporal derivative of first-order tensor fields, this is not a construction method for general 4D second-order tensors. The fact, that the partial derivative of the added dimension is always zero, allows us to utilize glyphs constructed in the remaining subspaces. This added dimension can then be projected onto the subspace and added to the glyph.

This work did not address deeper insights on visual perception of colors, controlled sampling of the underlying domain, or user studies, about the acceptance of the newly constructed glyphs. Dealing with cases of non-uniqueness when visualizing 3D tensors of rank 1 remains another inherited limitation of the glyph design based upon [3].

Our decision to move the sticks to the boundary of the glyph is mainly due to reducing visual clutter as well as to ensure visibility in the 3D case. However, in the 2D case, these sticks may give the impression to be only overlapped by the geometry and therefore be much longer, when the glyph is larger. As the two sticks represent the symmetry property of an eigenvector, their directions are identical and only reflected. They cannot, however, provide any information about which choice of sign represents the actual change of position of the feature to the next time step.

7 REFERENCES

[1] Rita Borgo, Johannes Kehrer, David H Chung, Eamonn Maguire, Robert S Laramee, Helwig Hauser, Matthew Ward, and Min Chen. Glyph-based visualization: Foundations, design guidelines, techniques and applications. *Eurographics State of the Art Reports*, pages 39–63, 2013.

[2] G. Farin. *Curves and Surfaces for CAGD*. Morgan Kaufmann, 5th edition, 2002.

[3] Tim Gerrits, Christian Rössl, and Holger Theisel. Glyphs for general second-order 2d and 3d tensors. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):980–989, 2017.

[4] Tobias Günther, Christian Rössl, and Holger Theisel. Opacity optimization for 3d line fields. *ACM Trans. Graph.*, 32(4):120:1–120:8, July 2013.

[5] Marcel Hlawatsch, Philipp Leube, Wolfgang Nowak, and Daniel Weiskopf. Flow radar glyphs& static visualization of unsteady flow with uncertainty. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):1949–1958, 2011.

[6] A. Kratz, C. Auer, M. Stommel, and I. Hotz. Visualization and analysis of second-order tensors: Moving beyond the symmetric positive-definite case. *Computer Graphics Forum*, 32(1):49–74, 2013.

[7] Robert S Laramee, Gordon Erlebacher, Christoph Garth, Tobias Schafhitzel, Holger Theisel, Xavier Tricoche, Tino Weinkauf, and Daniel Weiskopf. Applications of texture-based flow visualization. *Engineering Applications of Computational Fluid Mechanics*, 2(3):264–274, 2008.

[8] Tony McLoughlin, Robert S Laramee, Ronald Peikert, Frits H Post, and Min Chen. Over two decades of integration-based, geometric flow visualization. In *Computer Graphics Forum*, volume 29, pages 1807–1829. Wiley Online Library, 2010.

[9] Armin Pobitzer, Ronald Peikert, Raphael Fuchs, Benjamin Schindler, Alexander Kuhn, Holger Theisel, Krešimir Matković, and Helwig Hauser. The state of the art in topology-based visualization of unsteady flow. In *Computer Graphics Forum*, volume 30, pages 1789–1811. Wiley Online Library, 2011.

[10] Thomas Schultz and Gordon L Kindlmann. Superquadric glyphs for symmetric second-order tensors. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1595–1604, 2010.

[11] Nicholas Seltzer and Gordon Kindlmann. Glyphs for Asymmetric Second-Order 2D Tensors. *Computer Graphics Forum*, 2016.

[12] H Theisel and HP Seidel. Feature flow field. In *Proceedings of the symposium on Data visualisation*, volume 2003, 2003.

[13] Markus Uffinger, Filip Sadlo, and Thomas Ertl. A time-dependent vector field topology based on streak surfaces. *Visualization and Computer Graphics, IEEE Transactions on*, 19(3):379–392, 2013.

[14] T. Weinkauf, H. Theisel, and O. Sorkine. Cusps of characteristic curves and intersection-aware visualization of path and streak lines. In *Proc. TopoInVis*, April 2011.

[15] Tino Weinkauf, Jan Sahner, Holger Theisel, and Hans-Christian Hege. Cores of swirling particle motion in unsteady flows. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1759–1766, 2007.

A Modular Approach Based on Graph Transformation to Simulate Tearing and Fractures on Various Mechanical Models

Fatma BEN SALAH, Hakim BELHAOUARI, Agnès ARNOULD and Philippe MESEURE
University of Poitiers
Laboratory XLIM/ASALI , UMR CNRS 7252,
86962, Futuroscope Poitiers, France
fatma.ben.salah, hakim.belhaouari, agnes.arnould, philippe.meseure @univ-poitiers.fr

ABSTRACT

This paper introduces an extension of a general framework that allows the simulation of various mechanical models (discrete or continuous ones, for different kinds of meshes, in any dimension). This framework relies on a topological model and a rule-based language, that performs sub-graph matching and, possibly, transformations. This extension allows topological modifications such as tearing and fractures for all the implemented physical models. A general process has been used to simulate fractures and tearing: the topological transformation is described using the provided rule-based language and its application is triggered when a selected criterion is verified. Several criteria are proposed, that depend upon the strain or stress generated by a single or a set of interactions. This method raises the question of the link between the location where a criterion is applied and the mesh elements involved in a modification. This question has motivated us to design new criteria which are closely related to the mesh modification. Using this method, a minimal number of mechanical data need to be updated after a transformation and any interaction relying on mesh features (vertex, edge, face, volume) that are suppressed or split can be automatically ignored.

Keywords

Physical simulation, tearing, fracture, criterion, topological model, graph recognition.

1 INTRODUCTION

Due to their complexity, tearing and fracture are ones of the most studied phenomena in computer graphics, especially in physically-based animation. They occur due to the stress undergone by an object when it deforms. Much work in the literature has studied these phenomena using physical models, as they produce more realistic and accurate results. These physical models can be discrete (e.g. mass/interaction) [NTB+91, HTK00, SWB00], or based on continuous mechanics and solved by a Finite Element Methods (FEM) [OBH02, OH99, MMD+01, KLB14]... In this paper, we consider tearing and fracture as similar phenomena, depending on the rigidity of the material of the simulated object.

To initiate the tearing/fracture in an object, a criterion is required. Several types of criteria have been defined in the literature. These criteria can be based on strain or deformation (for instance, when the length of a spring goes beyond a threshold [NTB+91]). Or criteria can be based on stress, that is forces applied by one or a set of interaction (for instance the separation tensor defined by [OH99]).

After selecting a criterion that initiates the tearing/fracture and, more precisely, defines its location, a topological transformation is applied. Many types of topological transformations have also been proposed in the literature. For example, splitting several faces/edges surrounding a vertex [SWB00], removing an element from the mesh [NTB+91], etc. These mesh modifications have an effect both on mass distribution and local interactions. Therefore, after a transformation, mechanical information has to be updated.

Previous work did not focus on the relation between the location of a selected criterion and the location of the topological modifications. However, it is seldom the same location, and the link between these two locations is not necessarily immediate. For instance, in [NTB+91], when a spring is too much stretched, the support edge is not removed alone, but also volumes around this edge.

In [BBAM17], a general mechanical framework using a topological structure and based on rules of graph transformation for physical simulation is defined. This paper proposes an extension of this framework that allows tearing/fracture while still preserving its genericity. More precisely, this extensions:

- Offers many types of criteria (proposed in other approaches) to initiate topological modifications applied in 2D and 3D to different physical models and different types of elements as well as many types of topological modifications;
- Formalizes the link between the locations of a criterion and a topological modification by using a graph transformation-based approach. New criteria that are more closely related to topological transformation are also proposed;
- Allows topological modifications with minimal update of mechanical information.

The paper is organized as follows: in Section 2, some previous work related to fracture and tearing is briefly presented. Section 3 details the used topological model, the modeling approach used by the framework described in [BBAM17]. Section 4 discussed the usually used topological modifications and how they have been hosted in our framework. Section 5 presents the simulation of tearing/fraction using different criteria. Finally, some results and a discussion are given in Section 6.

2 PREVIOUS WORK

Topological modifications of physical models have been widely studied. Many approaches have simulated the cutting of deformable bodies, often for medical applications [WWD15]. However, cutting is a user-controlled phenomenon that does not require a criterion to trigger it. In practice, only the contact between the body to cut and an appropriate tool is sufficient to initiate cutting.

Although some approaches such as [LBC$^+$14] have focused on tearing and/or fracture using meshless models, most methods actually rely on meshed objects. Contrary to cutting, tearing and fracture require a criterion to determine whether a crack should appear or not, the location of this crack and its direction. Thus, many types of criteria have been proposed in the literature. These criteria can be based on local strain or stress in an object, and can be subdivided into two categories, namely *Atomic criteria* and *Cumulative criteria*.

Atomic criteria focus on a single interaction and consider a force or a deformation threshold to trigger a topological modification. For instance, Norton et al. [NTB$^+$91] use a mass/spring system (MSS) to simulate deformation and check if the length of springs exceeds a given value to decide where a fracture should be applied. The same criterion has been used by many following approaches: Hirota et al [HTK00] to simulate fracture on the surface of drying clay, Boux de Casson et al. [BL00] to simulate tearing of biological tissues represented by triangular meshes in 2D and tetrahedral meshes in 3D. It is also been used in [DKS$^+$11] and [LBC$^+$14].

Using cumulative criteria, a tearing/fracture is triggered if the sum of internal forces applied on a vertex or an element exceeds a threshold. Most methods that have used this type of criteria are based on continuous mechanics. The pioneer approach has been proposed by O'brien et al. for brittle materials [OH99] and ductile ones [OBH02]. Their criterion is based on a separation tensor computed for each vertex that triggers a fracture when one of its eigenvalues exceeds a given threshold. Recently, Koshier et al. [KLB14] used a similar criterion, but a tensor is computed for each vertex as the average of all stress tensors of its surrounding elements. Many other studies rely on a vertex tensor in 2D or 3D [IO09, WRK$^+$10, PNdJO14, PO09].

All above-mentioned approaches use various topological transformations. Norton et al. [NTB$^+$91] remove one or more elements. Faces or volumes are split in [SWB00] and [BL00], elements are cut and duplicated in [MBF04]. In [OH99, OBH02, KLB14], for a vertex to split, adjacent volumes are separated (by splitting a fan of faces) in 3D. In 2D, two faces must be separated by splitting their common edge. To facilitate this step, some studies use a predefined crack pattern [IO09, MCK13, PO09]...

Note that the link between the criterion location and the resulting topological transformation is not necessarily immediate. For instance, in [DKS$^+$11], when the length of a spring exceeds a threshold, one of its two extremities is arbitrarily split. In [NTB$^+$91], the same criterion leads to an element removal, even if all the other springs of the element do not exceed the deformation threshold. Since no solution to directly deal with the support edge of the spring has been proposed, some consecutive methods have chosen not to rely on a mesh [LBC$^+$14]. Any criterion that should result in a vertex split, often requires to split a fan of faces in 3D and one or two edges in 2D. Finally, no approach focuses on the relation between the location of the fracture criterion and the location of the consequent topological modification.

After any topological modification, some mechanical properties (for instance, mass or stiffness) have to be updated. Using FEM, only the mass of vertices have to be updated, which can however be a tedious process, depending upon the method used to take inertia into account. Using MSS, this step can be considered as costly [FZDJ14, MDS10], because springs split into

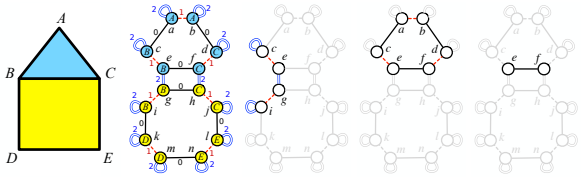
several elementary springs. To summarize, the application of a topological modification can be divided into three steps: first the choice of a criterion, second, a topological modification to apply and, last, the update of mechanical information.

Some general frameworks that allow objects simulation using various physical models exist (Sofa is probably the most general one [Sof]). However, no general frameworks that allow the tearing and/or fracture with many types of criteria and topological modifications have been proposed. The approach proposed in this paper relies on the general framework described in [BBAM17] which is based on rules of graph transformations applied on a topological model. This framework seems to be the most adapted to our needs due to the fact that it is multi-element, multi-physics and multi-dimension. Furthermore, all mechanical properties and forces are stored at convenient topological places. To generate a modular system for tearing and fracture, several types of criteria (including new ones) and topological modifications have been integrated in this framework.

3 MODELING AND SIMULATION

3.1 Topological Model

Much work have shown the benefits of using a topological model to simulate topological transformations of an object [MDS10, FZDJ14]. The model used in [BBAM17] is the generalized maps (G-maps) [DL14]. This topological model is based on the concept of *dart*, that can be considered as the extremity of an edge of a face of a volume.



(a) (b) (c) (d) (e)

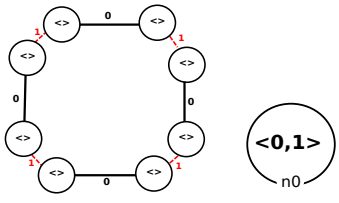
Figure 1: Decomposition of a 2D object and examples of orbits: (a) Object 2D, (b) G-map (connected component), (c) Vertex, (d) face and (e) half edge.

The representation of an object using a G-map is defined from its successive subdivisions into topological cells (volumes, faces, edges...). For example, the 2D object presented in Fig.1a can be decomposed into 2-dimensional cells (faces) connected by 2-links (blue double arcs). The faces are also split into 1-dimensional cells (edges) connected by 1-links (red arcs). In the same way, edges are decomposed into 0-dimensional cells (darts) connected by 0-links (black arcs). As represented in Fig. 1b, a G-map can be defined as a graph where nodes are darts and arcs labeled in $0,..,n$ are the adjacency relationships between topological cells (vertice, edge, face...).

These cells are defined by sub-graphs called *orbits*. Fig. 1 presents some examples of orbits corresponding to cells. Thus the sub-graph reachable from the dart e using 1- and 2-links in Fig. 1c represents the vertex B of Fig. 1a. It is noted $\langle 1,2\rangle(e)$. Similarly, the edge BC is the orbit $\langle 0,2\rangle(e)$, and the face ABC is represented by the orbit $\langle 0,1\rangle(e)$ (Fig. 1d). There exists also orbits which are not cells, for example the half-edge in Fig. 1e, or the connected component (Fig. 1b).

Depending on the targeted application, to model an object , much information (geometrical, mechanical, colorimetric...) must be added to the topological structure. These data are carried by all the darts and are called *embeddings*. Note that every embedding has its own data type and is attached to a particular orbit. For example in Fig. 1b, all the darts of the same vertex (A, B, C and D of Fig. 1a) are supplied with the same position information. Similarly, all darts of every face orbits $\langle 0,1\rangle$ share the same color information (blue or yellow). Evenly, information can be associated to any type of orbits such as corner of face $\langle 1\rangle$.

In this paper, an approach based on rules of graph transformation is used. These rules are created using a tool called Jerboa [BALB14], freely available at [Jer]. A rule in Jerboa is composed of two members and has the following form $L \rightarrow R$. The left member L contains the matched pattern that has to be recognized in the structure. The right member R corresponds to topological and/or embeddings modifications of this matched pattern. In fact, the application of a rule on a graph G consists in searching the sub-graph presented in the left part and replace it by the sub-graph presented in the right member of the rule. If the structure does not contain a matched pattern, the application of the corresponding rule fails.



(a) Rectangular face (b) Face

Figure 2: Examples of matched pattern

A matched pattern can be represented explicitly. In this case, all nodes have to be labeled by the dart orbit ($\langle\rangle$). An example of a rectangular face with all its darts is represented on Fig. 2a. A matched pattern can be also represented implicitly using darts labeled by orbits. For instance a face $\langle 0,1\rangle$ is represented in Fig. 2b. This pattern can match not only rectangular faces but all possible types of faces (rectangular, triangular,...). Some examples of rules are presented in section 4.

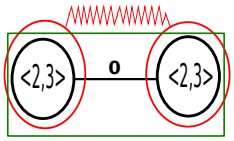
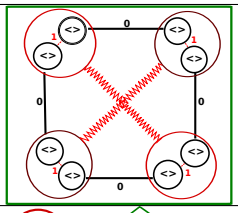
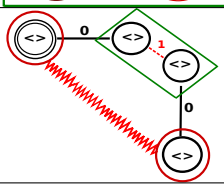
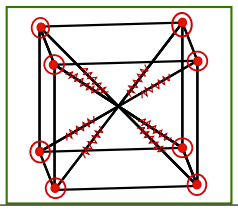
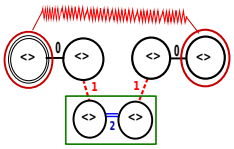
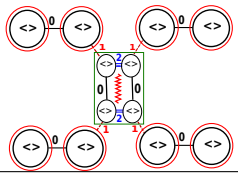
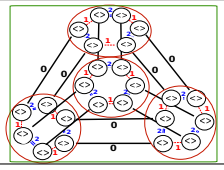
Interaction	*Matched pattern*	*Interac -tion orbits*	*Forces orbits*
Stretching spring		Half-edge: $\langle 0 \rangle$	Dart: $\langle\rangle$
Linear shear spring		Face: $\langle 0,1 \rangle$	Face corner: $\langle 1 \rangle$
Angular shear spring		Face corner: $\langle 1 \rangle$	Dart: $\langle\rangle$
3D shear spring		Volume: $\langle 0,1,2 \rangle$	volume corner: $\langle 1,2 \rangle$
Linear bending spring		Edge extremity: $\langle 2 \rangle$	Dart: $\langle\rangle$
Angular bending spring		Edge: $\langle 0,2 \rangle$	Dart: $\langle\rangle$
Co-rotational		Volume: $\langle 0,1,2 \rangle$	Volume corner: $\langle 1,2 \rangle$

Table 1: Modeling of interactions

3.2 Modeling

The framework described in [BBAM17] has been used. It is multi-dimensional (2D,3D), multi-element (allowing triangular, rectangular, hexahedral, etc. elements) and multi-physics (mass/spring, mass/tensor, finite element models). It represents objects by a 2D/3D mesh and stores as specific embeddings any mechanical data and forces needed to simulate them. Actually, it is based on G-maps and rules of graph transformation presented in previous section. In this approach, the mass is distributed between particles that are placed on each vertex of the mesh. To compute forces applied by interactions, the same method is defined for all mechanical models and for all types of meshes. First, every data characterizing the interaction (stiffness, damping,...) is carried by an orbit. This orbit is related to the source/origin of the interaction and is represented by a green frame in Table 1 and more specifically in the third column. From this orbit, a matched pattern is defined. This sub-graph represents the left member of the rules used to compute forces. It allows the determination of the vertices involved in the forces computation (that contribute to the calculation and/or undergo forces). Finally, these forces are stored in sub-orbits of each involved vertex' orbit. They are represented in a red frame in Table 1 and explained in the last column.

The first lines of Table 1 presents MSS with the three types of springs defined by Provot [Pro95] (stretching, shear and the linear bending). Angular springs are also defined to control the angle between two adjacent faces [GHDS03]. In 3D, shear springs are also modeled. The last line in Table 1 presents how the FEM based on co-rotational [MG04] is modeled. Note that for continuous models, it is compulsory to tag a dart as the first dart to define the order of vertices and respect this order while exploiting the element stiffness matrix and its stress tensor.

3.3 Simulation

The simulation loop used in [BBAM17] to simulate objects is the same for all physical models. (see Fig. 3). The loops begins by computing the forces applied to each particle. This is done in three steps. First, by walking through the structure, any orbit that embeds properties of an interaction is found. Second, if this interaction's matched pattern is recognized, the involved vertices are identified. Third, the interaction rule is applied that is, the interaction is calculated and the obtained forces are embedded in the planned orbits. After this process, the algorithm computes the acceleration depending on Newton's second law. For this purpose, it walks through the structure and, for every vertex, collect forces stored in its sub-orbits($\langle 2 \rangle$,$\langle 1 \rangle$,$\langle 0 \rangle$,$\langle 1,2 \rangle$...). After that, it computes new velocities and positions of particles. Finally, it walks through the structure, evaluates the chosen criterion, and when the condition is satisfied, applies the corresponding topological modification.

A special attention must be paid to the storage of particles' mass. In practice, the mass of a particle comes from faces/volumes surrounding the corresponding vertex. Therefore, an embedding called "corner mass" is stored in the orbit corner of face (2D) or volume(3D) ($\langle 1 \rangle$ or $\langle 1,2 \rangle$), to memorize the contribution of each face/volume surrounding a vertex. Moreover, to allow

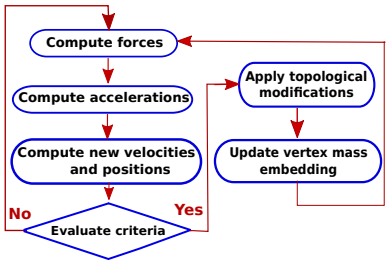


Figure 3: Fracture plan.

a fast computation of a particle behavior, its total mass must be known and is also stored in a dedicated embedding, in its corresponding vertex orbit. It is called "vertex mass" and defined as the sum of all masses stored in all its corner sub-orbits. In case of topological modification, this last embedding must be updated.

4 TOPOLOGICAL MODIFICATIONS FOR TEARING/FRACTURE

Topological modifications such as tearing/fracture can be simulated very easily using G-maps. More complex modifications can thereafter be built using a composition of separations of adjacent elements. For instance, it is possible to isolate an element and to remove it [MDS10]. It is also possible to separate several faces or volumes, to make a vertex split. In this section, we first focus on separation of elements, since it is a fundamental modification, before describing modifications based on composition of transformations.

4.1 Separation of Adjacent Elements

In 2D, two adjacent faces can be separated by splitting their common edge, more precisely by removing the 2-links that bind them as shown in Fig. 4. In a similar way, two adjacent volumes can be separated by splitting their common face, that is, removing the 3-links that bind them, as shown in Fig. 5. These link removals are elementary transformations in G-maps, that automatically deals with vertex splits [MDS10]. Note that, in 2D, removing 2-links should cancel the corresponding bending interactions (linear or angular). Using the rule-based approach, any suppression of topological links can automatically make any interaction be ignored, if the support orbit of the interaction embedding includes these links. Indeed, since the associated pattern no longer matches the local, unlinked, structure, the rule can not be applied anymore. The rule just has to reset all the corresponding forces.

Shear springs are stored in orbits $\langle 0,1 \rangle$ or $\langle 1 \rangle$, and they are not influenced by the removing of 2 and 3-links. Neither do stretch springs stored in orbits $\langle 0 \rangle$. Concerning continuous mechanics, in $2D$, no embedding are supported by orbits with 2-links. In $3D$, orbits of embeddings do not include any 3-links. Finally, no embedding needs to be changed. Only *vertex mass* embedding of split vertices must be updated.

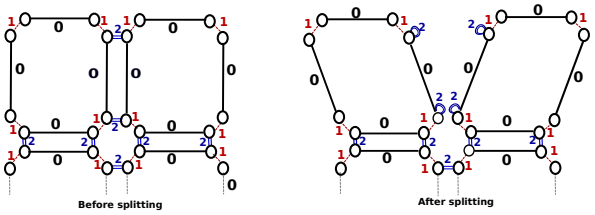


Figure 4: Edge split to separate two adjacent faces.

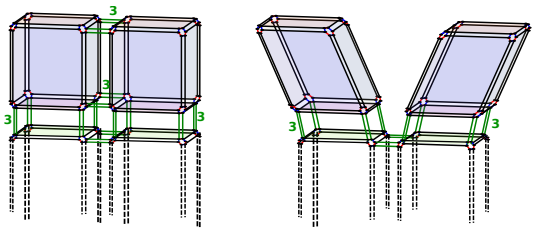


Figure 5: Face split to separate two adjacent volumes.

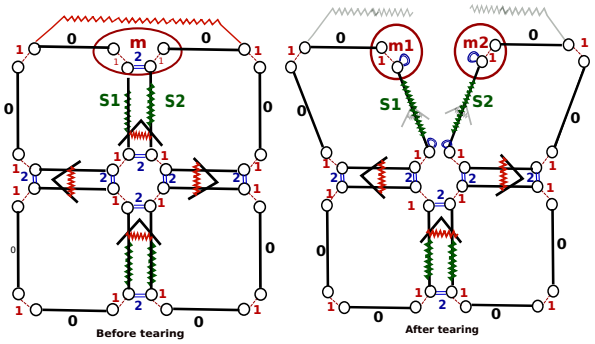


Figure 6: Impact of tearing/fracture on mass embedding and angular springs.

For instance, in Fig. 6, if adjacent faces are separated, 2-links are broken and the common edge is split. As a result, bending springs (angular or linear one) embedded in sub-orbits $\langle 2 \rangle$ or $\langle 0,2 \rangle$ of the split edge are ignored after the separation. More precisely, while identifying the sub-graph corresponding to the force computation, the pattern does not match, so forces can not be computed. This example also explains that the accumulation embedding *vertex mass* stored in vertices (orbits $\langle 1,2,3 \rangle$) has to be updated as it relies on 2-links that have been suppressed. In fact, as shown in Fig. 6, the *vertex mass* is the contribution of two *corner masses* provided by the two adjacent faces ($m_{vertex} = m1 + m2$). After the edge split, this vertex splits and each new vertex gets a new *vertex mass* value, (computed, here, using a single *corner mass*).

Rules for Separation of Elements

To split a face between two volumes, the rule presented in Fig. 7 has been created. In the left member of this rule, two faces are bound with 3-links. These faces are represented by their orbits $\langle 0,1 \rangle$. In the right member, the 3-links are broken to split the two faces. However this rule only copes with topology and does not handle embeddings (they are handled in a dedicated rule).

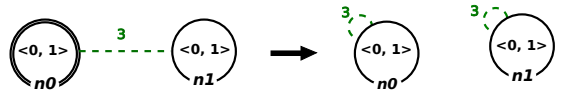


Figure 7: Rule to separate two adjacent volumes.

Another rule is presented in Fig. 8. It has been created to separate two adjancent faces by splitting an edge. In this rule, the pattern presented in the upper part of the figure matches explicitly the 2-links between two adjacent faces. The lower part of the figure describes the topological modification which is the suppression of 2-links between faces. As linear bending springs are stored in orbits $\langle 2 \rangle$ and angular ones in orbits $\langle 0,2 \rangle$, they are removed when the 2-links are broken. This single rule updates simultaneously the embedding *mass vertex*, the topological modification and resets the forces applied by the bending springs. When a vertex splits, the other embeddings (position, velocity, etc.) are automatically duplicated so the new particle is taken into account in the forthcoming steps of the simulation.

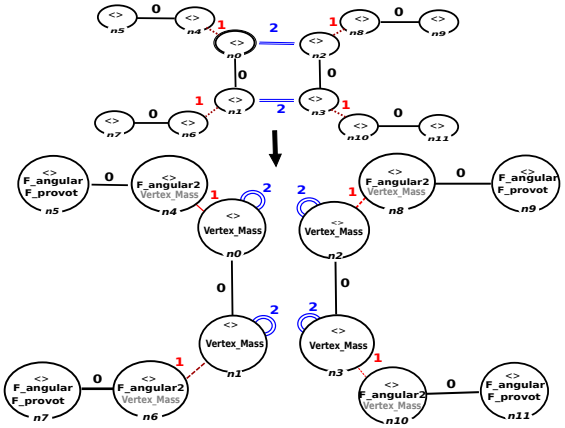


Figure 8: Rule to separate two adjacent faces.

4.2 Mesh Element Removal

To suppress an element (a volume in 3D and a face in 2D), the same process described in [MDS10] is used. This process consists first in the isolation of the element: in 2D, all its edges are 2-unlinked, in 3D, all its faces are 3-unlinked. This separation can be done using the rules described in Section 4.1. The second step consists in deleting the isolated element (using a rule with an empty right member).

4.3 Vertex Split

As proposed in [OH99], to split a vertex, a fan of faces surrounding this vertex can be split. In 2D, only two edges have to be split. A tearing/fracture plane must be computed based on the selected criterion. Then, the sign of all elements surrounding a vertex is determined by replacing the coordinates of the center of each element in the plane equation. The sign is stored in the orbit of every element as an embedding. All edges/faces placed between two elements with different signs have to be split using rules described in Section 4.1. This topological modification can be applied for any interaction model (MSS, FEM).

4.4 Edge Removal

If a criterion relies on an edge, it can require that this edge disappears. However, removing an edge from a mesh is a tricky topological modification. Indeed, this removal can lead to a type change of surrounding elements. For instance, the contraction of an edge in a rectangle mesh can induce a transformation of elements into triangles. Therefore, as a solution, it is better to use more general transformations involving the concerned edge that also control adjacent elements. For example, one or more elements adjacent to this edge can be deleted, as proposed by Norton et al. [NTB+91]. This modification is based on element removal described above (see Section 4.2). Vertex split can also be used [DKS+11]. Finally, whatever the chosen transformation, this last influences not only the used criterion's cell but also many adjacent mesh elements.

5 SIMULATION OF TEARING/FRACTURE

Tearing or fracture occurs when a given criterion is met. Different criteria exist and have been added to the framework. These criteria act as classical preconditions used by Jerboa that trigger topological modifications (see Section 4) when they are satisfied. As explained in Section 2, criteria can be divided into two categories: atomic and cumulative ones. Concerning cumulative criteria, additional rules must be created to compute the sum of needed stresses/forces.

5.1 Atomic Critera

Atomic criteria only relate to a single interaction. Criteria are based on strain or stress considerations. Note that stress-based criteria are particularly well adapted to our models, since, as described in section 3.2, all forces applied by all types of interactions are stored in sub-orbits of vertex orbits. Concerning linear (resp. angular) springs, strain or stress criteria are equivalent [BL00], that is, a length (resp. angle) or a force (resp. torque) threshold can be checked. If the chosen threshold is exceeded, the process consists in removing the spring. To remove a spring, the required topological transformation depends on its type. Stretch springs are placed on edges. As discussed in Section 4.4, volumes can be removed, as proposed in [NTB+91, HTK00, BL00]. If the same criterion is checked on a shear spring, the support element (volume in 3D or face in

2D) should be suppressed. If a bending (linear or angular) springs verifies the tearing/fracture criterion, the two concerned faces must be separated, as described in Section 4.1. In a similar way, if continuous mechanics is used, eigenvalues of the stain or stress tensor of each element can be computed and, if a threshold is exceeded, the concerned element is suppressed.

However, to avoid the design of specific rules for all kinds of interaction, it is possible to use a more general approach, that can be applied both in 2D and 3D. Indeed, a criterion can be defined by checking if a force, whatever its origin, exerted on a vertex, exceeds a threshold. By considering a fracture plane perpendicular to this force, a vertex split transformation can be applied (Section 4.3). Note that, due to the action/reaction principle, the two extremities of a spring or a set of involved vertices for other interaction types are likely to be concerned by a threshold exceeding. In this case, it is possible to apply a vertex split on all the concerned vertices (to simulate multiple cracks) or on a single (arbitrarily chosen) vertex [DKS+11].

5.2 Cumulative Criteria

Cumulative criteria first compute the sum of all forces/stresses applied on every vertex. Using continuous mechanics, this method corresponds to the O'Brien et al.'s approach [OH99]. However, the simplified approach proposed in [KLB14] is used in our framework. The stress tensor (expressed here as a 6-vector in 3D) of each element i is computed as explained in [MDM+02]:

$$\sigma_{\mathbf{i}} = (\mathbf{C} * \mathbf{B_i})\mathbf{u} \tag{1}$$

With $\mathbf{C}$ the stress-strain matrix, $\mathbf{B_i}$ is the strain-displacement matrix (for more detail the reader must refer to [MDM+02]) and $\mathbf{u}$ is the displacement vector. A vertex tensor is computed by averaging the stress tensor of surrounding elements i:

$$\sigma_{\mathbf{vertex}} = \sum m_i \sigma_{\mathbf{i}} / \sum m_i \tag{2}$$

With m_i the mass of the element i and σ_i, the 3×3 stress tensor of this element. The eigenvalues of the vertex tensor are computed. If one of these values exceeds a threshold, the corresponding eigenvector is computed. Finally, the fracture plane is defined as perpendicular to this eigenvector. A vertex split is applied as described in Section 4.3. A similar approach can be used in 2D.

A more general cumulative criterion can be proposed, whatever the used deformation laws. Using a walk through all the forces stored in every vertex orbit (see Table 1), the sum of internal forces can be computed for each vertex. When the magnitude of such a force exceeds a threshold, a vertex split is triggered using a fracture plane perpendicular to this force. Note that this kind of criterion can only work in particular cases, for instance when internal forces must compensate a high external stress, or when a sudden deformation appears locally in the mesh (after the motion of one or more vertices). On the contrary, at rest, the internal forces can be null and no tearing/fracture appears, even if atomic interactions produce high magnitude forces.

5.3 Criteria dedicated To Mesh Elements Separation

As stated in [MDS10], separation of elements is the fundamental topological modification that is required to represent tearing or fractures, since it is the core on which the other transformations are based. However these transformations are not enough local and involve multiple elements not directly concerned by the used criterion. Separations of elements are surely desirable, but, unfortunately, no above-mentioned criterion results in a single application of such elementary modifications. In a similar way as Smith et al.[SWB00] who proposed a constraint-based criterion to split a face between two volumes, force-based solutions are investigated to trigger element separation. On other words, we want to find criteria that are applied on a cell and that trigger a modification on this same cell.

In 2D, faces are separated by splitting an edge, so this edge should support the fracture criterion. More precisely, the forces applied to the edge AB by all interactions supported by the adjacent faces are compared to check if they tend to make the edge split. First of all, the direction $\mathbf{n}$ along which the forces are compared (see Fig. 10) is computed as:

$$\mathbf{n} = \mathbf{AB} \times (\mathbf{n_1} + \mathbf{n_2}) \tag{3}$$

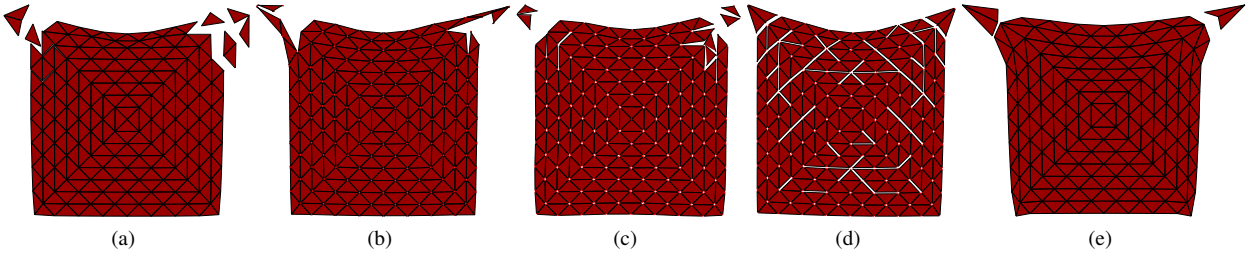
Let $\mathbf{f_A^{(i)}}$ and $\mathbf{f_B^{(i)}}$ the forces applied by each adjacent faces i on vertices A and B. These forces just require to walk through the orbits of A and B restricted to each face (orbit $\langle 1 \rangle$ in 2D, orbit $\langle 1,2 \rangle$ in 3D) and collect all calculated forces stored in sub-orbits. Let $\mathbf{f_i} = \mathbf{f_A^{(i)}} + \mathbf{f_B^{(i)}}$, the forces applied on the edge. A possible criterion compares the effect of these forces by computing:

$$|(\mathbf{f_1} - \mathbf{f_2}) \cdot \mathbf{n}| \tag{4}$$

If the magnitude of this force is beyond a threshold, and if each force tends to contract the elements, then the common edge can split. This occurs when forces $\mathbf{f_1}$ and $\mathbf{f_2}$ are along opposite directions, and at least one of them has a high magnitude. This criterion is called "atomic edge criterion".

Let $\mathbf{f_A}$ and $\mathbf{f_B}$ the overall forces (coming from any element) applied on these vertices. Another criterion consists in computing the magnitude of the overall forces applied on the edge, projected on $\mathbf{n}$, and comparing it to a given threshold:

$$|(\mathbf{f_A} + \mathbf{f_B}) \cdot \mathbf{n}| \tag{5}$$



(a) (b) (c) (d) (e)

Figure 9: Examples of tearing a 2D object with different criteria.

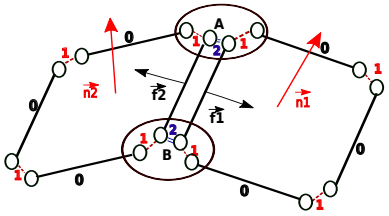


Figure 10: Computing internal forces of an edge.

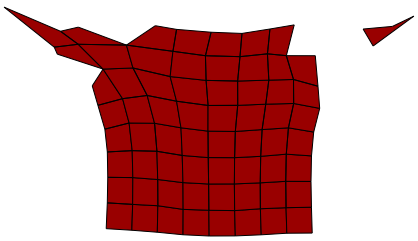


Figure 11: Edge split with criterion based on the difference between forces applied on an edge.

This kind of criteria is called "global edge criterion" and can also be extended as a face criterion in 3D. In this case, the normal of the face to split **n** is used to project the different faces and compute their contribution to the face split. As above, only forces stored in the volume orbits are first collected, projected on **n** and are compared to check if the common face splits. The overall forces applied to the face vertices can also be used to provide another criterion.

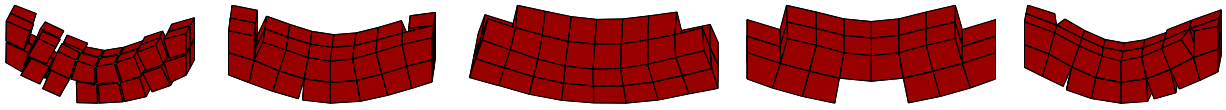
6 RESULTS AND DISCUSSION

We have implemented the criteria and topological modifications cited in the previous sections using the rule-based language. Thus, Fig. 9 presents the same object, a cloth modeled as a 2D triangular mesh, simulated with different types of criteria. Its two upper extremities are always fixed. In Fig. 9a, this cloth is modeled with co-rotational FEM and is torn by vertex split when the force applied by at least one of its adjacent face is beyond a threshold (atomic criterion). Fig. 9b presents the same cloth modeled as MSS, but when the force generated by a spring exceeds a threshold, its extremities split. Then, Fig. 9c presents this same system where a vertex splits wherever the sum of applied forces exceeds a threshold. In Fig. 9d, the same model is used, and an edge of the mesh can split where the *global edge criterion* is verified. Finally, Fig. 9e still presents the same MSS with edge splits where the *atomic edge criterion* is satisfied. As can be seen, different fracture criteria produce completely different results, where tearing is concentrated on the constraint areas or is disseminated in the overall body.

Similar simulations can be applied on rectangular or mixed meshes using the same criteria and topological modifications. An example of such a rectangular mesh is presented in Fig. 11. Fig. 12 presents some other results of fractures in 3D using mass/spring system and different criteria. An example of tearing of a liver (consisting of 597 elements) simulated using the co-rotational method is presented in Fig. 13.

The performance of our model is not the main concern of this paper. Indeed, Jerboa is a tool dedicated to prototyping and not interactive simulation, even if it allows the simulation of topological transformations on complex meshes. As a consequence, some of the discussed models have been ported to C++ to measure their efficiency. This specific implementation strictly respects all the principles described in this paper. In particular, when an interaction embedding is found, the corresponding graph is used to find how the interaction is computed and where forces apply. We chose to focus on a 2D cloth model consisting of 10×10 rectangular patches that can be further triangulated to produce a local triangular mesh. An example of simulation using this optimized implementation is presented in Fig. 14.

Some simulation times are summarized in Table 2 for different type of meshes, using Runge-Kutta 4 integration to allow a more stable simulation. No parallelization method has been used. Note that, thanks to the ruled-based approach, the implementation allows the simulation of heterogeneous meshes, with a little computation over-cost (below 1%) compared to meshes consisting exclusively of quads or triangles. Simulation have been measured on a 2.7GHz Intel I7 system (3740QM Processor). All simulation times are actually compatible with real time. Simulating with quads is faster because this model is subdivided in less faces. The mixed mesh triangulates half the quads of the initial mesh. It appears as a real compromise between quad and triangle meshes. The global edge criterion is faster



(a) Atomic force criterion and vertex split (b) Sum of forces criterion and vertex split (c) Atomic forces criterion and volume suppression (d) Atomic force criterion of shear springs and edge suppression by removing volumes (e) Atomic force criterion applied on a face and face split

Figure 12: Examples of fracture of a 3D object with different criteria.

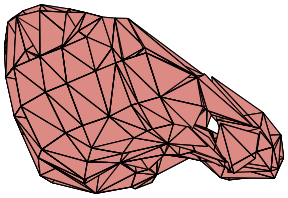


Figure 13: Elementary forces and vertex split (FEM).

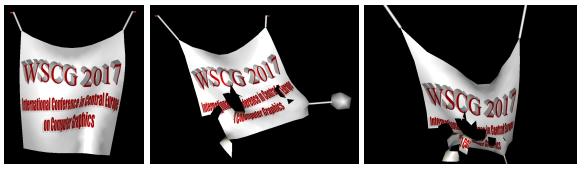


Figure 14: A cloth simulated in real-time (mass = 100g, stretching stiffness = 60 N/m, shear stiffness = 43 N/m, bending stiffness = 10 N/m, damping = 0.1, fracture threshold = 0.5 N), using a global edge criterion.

	Basic step	Global edge criterion	Atomic edge criterion
Quads	0.37	0.41	0.55
Triangles	0.78	0.95	1.3
Mix	0.61	0.72	0.97

Table 2: Simulation times (in ms) for a cloth depending upon the type of mesh. Basic step shows the simulation times without any tearing detection.

than the atomic one, because this last requires to store atomic forces in the model instead of only accumulating them in each vertex, which can be a costly process. However, this criterion triggers tearing only on points of very high stress (constrained particles in the example), which can be a desired behavior.

7 CONCLUSION/PROSPECTS

This paper proposes an extension of the framework presented in [BBAM17] which relied on rules of graph transformations and a topological model to simulate various deformable bodies in 2D and 3D. This extension provides this framework with topological transformations such as tearing and fractures. The proposed approach is modular, since the user can select a criterion (or more) and a suitable mesh modification. This makes it an interesting tool to control tearing in an animation/simulation system. Many known types of criteria are implemented, some of them are specific to a given mechanical model and others are completely general. To produce mesh modifications when a criterion is met, several types of topological transformations are proposed. However, some of these modifications (vertex split, edge removal) tend to involve a lot of adjacent elements and are surely not enough local, so often require to use a high resolution mesh or subdivide the elements locally. To circumvent this problem, we propose to use more closely-related criteria and topological modifications, that is, criteria and topological modifications based on the same topological cell (face in 3D, edge in 2D). The mechanical information of the model is stored in an atomic way, therefore no update of interaction properties such as spring stiffness is needed after a topological change. The same way, any interaction that depends on the modified cells is automatically ignored thereafter. Only mass of particles must be updated, but this process consists in gathering elementary masses stored in the associated vertex' orbit. Simulations based on different scenarios show the effectiveness of our approach.

We have investigated its efficiency through a C++ implementation of some 2D models. The measured simulation times allow interactive and real-time simulations. However, it is possible to get even better simulation times by combining some interactions that appear as redundant: For instance, all stretch springs corresponding to the same edge can be cumulated in a single spring (that is, stiffness and damping values are added) and resulting forces computed once instead of being computed for each atomic spring. In 2D, only two springs are to be cumulated, but in 3D, more atomic springs are generally involved. Other redundancies exist, in particular in 3D (shear springs of 3-linked faces, linear provot springs placed between the same vertices, etc.). However, this approach implies that some cumulated interaction properties should be updated after any topological change, and that some atomic forces will no longer be available to contribute to a tearing/fracture criterion. In future work, we want to include these optimization solutions to our framework, whenever the selected criterion allows it, and enhance it with other complex types of topological transformations, for instance subdivisions of elements or local remeshing .

8 ACKNOWLEDGMENTS

This work has been partially funded by the European Erasmus Mundus - Al Idrisi II scholarship program. It also received fundings from the MIRES federation.

9 REFERENCES

[BALB14] Hakim Belhaouari, Agnès Arnould, Pascale LeGall, and Thomas Bellet. Jerboa: A graph transformation library for topology-based geometric modeling. In *ICGT*, pages 269–284, 2014.

[BBAM17] Fatma Ben Salah, Hakim Belhaouari, Agnès Arnould, and Philippe Meseure. A general physical-topological framework using rule-based language for physical simulation. In *VISIGRAPP*, 2017.

[BL00] François Boux de Casson and Christian Laugier. Simulating 2D tearing phenomena for interactive medical surgery simulators. In *Computer Animation*, pages 9–14, 2000.

[DKS+11] Emmanuelle Darles, Saman Kalantari, Xavier Skapin, Benoît Crespin, and Annie Luciani. Hybrid physical topological modeling of physical shapes transformations. In *CASA*, 2011.

[DL14] Guillaume Damiand and Pascal Lienhardt. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing*. A K Peter/CRC Press, 2014.

[FZDJ14] Elsa Fléchon, Florence Zara, Guillaume Damiand, and Fabrice Jaillet. A unified topological-physical model for adaptive refinement. In *VRIPHYS*, pages 39–48, 2014.

[GHDS03] Eitan Grinspun, Anil N Hirani, Mathieu Desbrun, and Peter Schroder. Discrete shells. In *Symp. on Computer Animation*, 2003.

[HTK00] Koichi Hirota, Yasuyuki Tanoue, and Toyohisa Kaneko. Simulation of three-dimensional cracks. *Visual Computer*, pages 371–378, 2000.

[IO09] Hayley N. Iben and James F. O'Brien. Generating surface crack patterns. *Graphical Models*, pages 198–208, 2009.

[Jer] http://xlim-sic.labo.univ-poitiers.fr/jerboa.

[KLB14] Dan Koschier, Sebastian Lipponer, and Jan Bender. Adaptive tetrahedral meshes for brittle fracture simulation. In *Symp. on Computer Animation*, pages 57–66, 2014.

[LBC+14] Joshua A. Levine, Adam W. Bargteil, Christopher Corsi, Jerry Tessendorf, and Robert Geist. A peridynamic perspective on spring-mass fracture. In *Symp. on Computer Animation*, pages 47–55, 2014.

[MBF04] Neil Molino, Zhaosheng Bao, and Ronald Fedkiw. A virtual node algorithm for changing mesh topology during simulation. *Trans. on Graphics*, pages 385–392, 2004.

[MCK13] Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Real time dynamic fracture with volumetric approximate convex decompositions. *Trans. on Graphics*, pages 115:1–115:10, 2013.

[MDM+02] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *Symp. on Computer Animation*, pages 49–54, 2002.

[MDS10] Philippe Meseure, Emmanuelle Darles, and Xavier Skapin. Topology based physical simulation. In *VRIPHYS*, pages 1–10, 2010.

[MG04] Matthias Müller and Markus Gross. Interactive virtual materials. In *Graphics Interface*, pages 239–246, 2004.

[MMD+01] Matthias Muller, Leonard McMillan, Julie Dorsey, , and Robert Jagnow. Real time simulation of deformation and fracture of stiff materials. In *Works. on Animation and Simulation*, pages 113–124, 2001.

[NTB+91] Alan Norton, Greg Turk, Robert Bacon, John Gerth, and Paula Sweeney. Animation of fracture by physical modeling. *Visual Computer*, pages 210–219, 1991.

[OBH02] James F. O'Brien, Adam W. Bargteil, and Jessica K. Hodgins. Graphical modeling and animation of ductile fracture. In *SIGGRAPH*, pages 291–294, 2002.

[OH99] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. In *SIGGRAPH*, pages 137–146, 1999.

[PNdJO14] Tobias Pfaff, Rahul Narain, Juan Miguel de Joya, and James F. O'Brien. Adaptive tearing and cracking of thin sheets. *Trans. on Graphics*, pages 110:1–110:9, 2014.

[PO09] Eric G. Parker and James F. O'Brien. Real-time deformation and fracture in a game environment. In *Symp. on Computer Animation*, pages 165–175, 2009.

[Pro95] Xavier Provot. Deformation constraints in a masss/pring model to describe rigid cloth behaviour. In *Graphics Interface*, pages 147–154, 1995.

[Sof] https://www.sofa-framework.org/.

[SWB00] Jeffrey Smith, Andrew P. Witkin, and David Baraff. Fast and controllable simulation of the shattering of brittle objects. In *Graphics Interface*, pages 27–34, 2000.

[WRK+10] Martin Wicke, Daniel Ritchie, Bryan Matthew Klingner, Sebastian Burke, Jonathan Richard Shewchuk, and James F. O'Brien. Dynamic local remeshing for elastoplastic simulation. *Trans. on Graphics*, pages 49:1–49:11, 2010.

[WWD15] Jun Wu, Rüdiger Westermann, and Christian Dick. A survey of physically based simulation of cuts in deformable bodies. *Computer Graphics Forum*, pages 161–187, 2015.

Bandwidth and Memory Efficiency in Real-Time Ray Tracing

Pedro Lousada
INESC-ID/Instituto Superior Técnico, University of Lisbon
Rua Alves Redol, 9
1000-029 Lisboa, Portugal
pedro.lousada@ist.utl.pt

Vasco Costa
INESC-ID/Instituto Superior Técnico, University of Lisbon
Rua Alves Redol, 9
1000-029 Lisboa, Portugal
vasco.costa@ist.utl.pt

João M. Pereira
INESC-ID/Instituto Superior Técnico, University of Lisbon
Rua Alves Redol, 9
1000-029 Lisboa, Portugal
jap@inesc-id.pt

ABSTRACT

Real time ray tracing has been given a lot of attention in recent years in the academic and research community. Several novel algorithms have appeared that parallelize different aspects of the ray tracing algorithm through the use of a GPU. Among these, the creation of Bounding Volume Hierarchies (BVHs). We believe that recent approaches have failed to consider the performance impact of memory accesses in GPU and how their cost affects the overall performance of the application. In this work we show that by reducing memory bandwidth and footprint we are able to achieve significant improvements in BVH traversal times. We do this by compressing the BVH and the triangle mesh in a parallel manner after its creation, in each frame, and then decompressing it as needed while traversing the BVH.

Keywords

Ray-tracing, bounding-volume-hierarchy, parallelization, gpu, quantization, memory.

1 INTRODUCTION

Real-time rendering typically concerns itself with the generation of synthetic images at a rate fast enough that the viewer can interact with a virtual environment. As an image appears on screen, the viewer acts or reacts, and this feedback affects what is generated next. Two of the most popular approaches to synthetic image generation are Rasterisation and Ray-Tracing. Both have been used in computer graphics for the past decades. Each method allow us to generate 2D images from 3D scenes composed of virtual objects.

Real-time ray-tracing received little attention outside the academic world mainly due to its high computation costs which made it a much more expensive and slow approach compared to rasterisation. Ray tracing offers a fairly long list of advantages over rasterisation. Ray-tracing can easily simulate non-local effects such as shadows, reflections and refractions. In Rasterisation reflections and shadows are hard to compute; refractions are very hard.

Due to its mathematical correctness, ray tracing can make generated images look more realistic. The issue is being able to generate them at a rate fast enough as required by these real-time applications.

1.1 Problem

At its core, the ray tracing algorithm follows the following logic: for each pixel of the display, we cast rays that propagate in a straight line until they intersect an element of the scene being rendered. The color of the pixel is computed as a function of the material at the intersected element's surface, the incident light (radiance) function at the intersection point, and of the viewing direction (observer's position). For a simple scene with no secondary rays (i.e. reflections, shadows, refractions, etc) this means having at least NxM intersection tests (N being the number of rays and M the number of polygons in the scene). For its nature, ray tracing inherently leads to a high number of expensive floating point operations. This, together with an irregular and high bandwidth memory access pattern, means performance will be an issue when trying to achieve real time rendering.

One way to optimize the standard ray tracing algorithm is through the use of acceleration structures. Acceleration structures allow us to lower the number of intersections tests needed to render an image. Currently Bounding Volume Hierarchies (BVHs) are the most popular solution. The state of the art in this kind of structure

has been targeted towards generating the structure at a rate fast enough to be usable in a real time application, typically creating a new BVH or refitting an existing one at each frame.

Most modern algorithms use a GPU to achieve the performance they need. Despite their capability of performing a high number of floating point operations per second, GPUs suffer from slow memory access. Yet most algorithms that focus on constructing a BVH in parallel manner tend to overlook this and are careless with both memory accesses and bandwidth, resulting in poorer overall performance than otherwise possible.

1.2 Contributions

We believe that by optimizing GPU memory footprint and bandwidth efficiency we will be able to improve render performance. In this work we show how to develop a ray tracing application based on a state of the art algorithm in parallel BVH construction and then improve it further through memory compression techniques.

Our contribution is a novel algorithm, based on existing techniques for real time BVH construction, with focus on improvements in BVH and triangle mesh compression. We are able to reduce the total size of memory used to store both the BVH and the triangle mesh as well as reduce the memory bandwidth of the application. We saw improvements of up to 40% in occupied memory, and reductions of up to 20% in memory bandwidth, with our techniques.

2 BACKGROUND

First introduced by Whitted [Whi80], ray tracing is an algorithm for image synthesis where direct illumination (including shadows), and perfect reflections/refractions are simulated. It employs the use of ray casting to intersect eye rays with objects in a computer simulated scene. Eye rays which intersect objects lead to the creation of extra secondary rays: e.g. shadow, reflection and refraction rays.

The main difference between rasterisation and raytracing is the ability to simulate complex light effects such as shadows, reflections and refractions. Whereas rasterisation engines often simulate these effects through the use of texturing, ray tracing takes a more accurate analytical approach. Since these effects are highly geometry dependent, simulated methods can look unrealistic when changing object position or viewer position. The difference in ability to simulate secondary visual effects between rasterisation and ray tracing can be seen in Figure 1.

Figure 1: Rasterisation vs ray tracing (source: Intel)

2.1 Bounding Volume Hierarchies

One of the major problems of ray tracing is the sheer number of intersection tests one must perform in order to check if a certain ray intersects an object of the scene. Acceleration structures help us reduce the number of intersections to test by organizing the geometry of the scene into a data structure that can easily be explored. The organization of an acceleration structure is typically hierarchical, loosely meaning that the topmost level encloses the levels below it, and so on.

Bounding Volume Hierarchies (BVHs) are a tree-like structure that subdivides a scene into smaller portions. A BVH partitions a scene's objects. Each geometric primitive object is wrapped with an individual bounding volume. These form the leaf nodes of the tree. Bounding volumes are then recursively merged together until we are left with a single bounding volume wrapping the entire scene.

In a typical object hierarchy data structure it is easy to update the data structure as an object moves, because an object lives in just one node, thus the bounds for that node can be updated with relatively simple and localized update operations. For deformable scenes, just refitting a BVH - i.e., recomputing the hierarchy node's bounding volumes, but not changing the hierarchy itself - is sufficient to produce a valid BVH for the new frame. BVHs also allow for incremental changes to the hierarchy [WMG+09].

In BVHs, primitives are referenced exactly once, allowing us to save GPU memory and bandwidth. Empty cells, that frequently occur in spatial subdivision, do not exist in object hierarchies either. The effectiveness of a Bounding Volume Hierarchy for a particular scene depends on the characteristics of the hierarchy the build algorithm produces.

2.2 GPU Computing

Modern GPUs are SIMD (Single Instruction Multiple Data) devices [GPKB12], meaning that they can per-

form the same operation on multiple data points simultaneously. Even though a modern CPU core can dispatch more operations per second than a GPU core, the GPU as a whole can vastly outperform a CPU by having thousands of cores running the same operations versus the 4-8 cores present in a CPU. The massive parallelism of programmable GPUs lends itself to inherently parallel problems such as ray tracing. A given compute kernel executes a single program across many parallel threads. Typically, each kernel completes execution before the next kernel begins, with an implicit barrier synchronization between kernels. GPUs have support for multiple, independent kernels to execute simultaneously, but many kernels are large enough to fill the entire device. Threads are decomposed into thread blocks; threads within a given block may efficiently synchronize with each other and have shared access to per-block on-chip memory.

3 RELATED WORK

Generating an acceleration structure is a necessary step when trying to achieve real-time performances. One can take two approaches: To construct a new structure every frame or to reuse the same structure and adjust it at each frame. The latter option has been explored [KA13] but lacks performance when processing large trees or large modifications to the data structure are required.

Approaches that explore a construction of new hierarchy at each frame tended to rely on serial algorithms running on the CPU to construct the necessary hierarchical acceleration structures [GPM11]. While this was once necessary due to architectural limitations, modern GPUs provide all the facilities necessary to implement hierarchy construction directly. Doing so should provide a strong benefit, as building structures directly on the GPU avoids the need for the relatively expensive latency introduced by copying data structures between CPU and GPU memory spaces.

The first to explore such a method were Lauterbach et al. [LGS+09]. Lauterbach et al. introduced a novel algorithm using *spatial Morton codes* [Mor66] to reduce the construction of BVHs to a sorting problem. *Morton codes* are used to determine a primitive's order along a space filling curve. They can be computed directly from a primitive's geometric coordinates. The algorithm encloses each input primitive with an Axis-Aligned minimum Bounding Box (AABB) and determines the enclosing AABB of the entire input geometry. By taking the barycenter of each primitive's AABB as its representative point, and by quantizing each of the 3 coordinates of the representative points into k-bit integers, a $3k$-bit *Morton code* is constructed by interleaving the successive bits of these quantized coordinates. Figure 2 shows a 2D representation of this. Sorting the *Morton codes* will automatically lay the associated points in order along a *Morton spatial curve*. It will also order the corresponding primitives in a spatially coherent way. Because of this, sorting geometric primitives according to their *Morton code* is used to improve cache coherence since a ray that hits a certain primitive will likely also hit the primitive adjacent to it.

The main problem of this family of algorithms [LGS+09, PL10, GPM11] is that, in order to build the resulting BVH, a series of sequential steps must be taken. Lauterbach et al. create their BVH by sequentially observing each bit of each *Morton code* and grouping the primitives according to the value of the bit. At each level if said bit has value *0* then the primitive is placed in a group, if the bit has value *1* then it is placed in the opposite group. Garanzha et al. take a different approach by generating one level of nodes at a time, starting from the root. They then process the nodes of the BVH on a given level in parallel. For this they use binary search to partition the primitives contained within each node. The resulting child nodes are then enumerated using an atomic counter, and subsequently processed on the next round.

Karras et al. [Kar12] further developed this line of thought by describing an algorithm to build a BVH in a totally parallel manner. Karras et al. introduce an in-place algorithm for constructing a binary radix tree (also called a *Patricia* tree) which can directly be converted into a BVH.

Their approach is based on the fact that for a scene with N primitives we know we can make a *Patricia* tree with $N-1$ internal nodes to represent it. The similarities between each *Morton code* and its neighbours are analyzed to determine the position of each internal node in the Patricia Tree. The child-parent association is then calculated based on the range of *same-value-bits* with the rest of the *Morton codes*.

BVHs have a large memory footprint due to the need to store the bounding boxes. Hence several approaches have been used, over the CPU, to allow the visualization of large models without paging data from disk. Mahovsky et al. [MW06] and Bauszat et al. [BEM10] quantize the bounding box data, with significant memory and bandwidth savings, at the expense of extra computations. These approaches degrade rendering performance for models that fit uncompressed within main memory, even with a ray-bundle scheme, due to limited available CPU math performance. However a GPU is more bandwidth than math constrained so this conclusion needs to be revisited in that case.

4 APPROACH AND ARCHITECTURE

In order to achieve a high frame rate one must reduce the time it takes to generate an image at each frame.

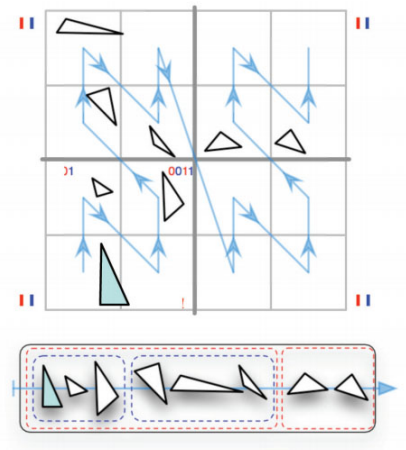


Figure 2: Example 2-D Morton code ordering of triangles with the first two levels of the hierarchy. Blue and red bits indicate x and y axes, respectively. (source: NVIDIA)

For this we make use of a GPU to help us accelerate all the floating point operations required to compute the intersections in a ray-tracer. We aim to reduce the memory bandwidth and memory footprint of our application. We make use of research, made in the area of parallel BVH construction in GPUs, to reduce the number of intersection tests performed (thus reducing memory bandwidth as well). We then explore BVH and triangle meshes compression techniques. With a compressed BVH and triangle mesh we expect our rendering kernels to achieve better rendering times since the data transferred between the GPU's global memory and the kernel's local memory will be smaller.

This is described in further detail in the following sections.

4.1 Binary Radix Tree Properties

Before we describe our algorithm there are a few Binary Radix Tree properties we should cover as these are important to understand our work.

A radix tree is a space-optimized tree often used for indexing string data, although it can also be used to index any data divisible in smaller comparable chunks such as characters, binary numbers, etc. For simplicity, assume that from now on our radix tree only contains binary values and that they are in a lexicographical order as in our application each key will correspond to a sorted *Morton code*.

Given a set of keys $k_0, ..., k_{n-1}$ represented as bits, a radix tree can be seen as a hierarchical representation of the common bits of each key. The keys are represented in the leaf nodes of the tree, and each internal node corresponds to the longest common prefix shared between the keys in that subtree.

Assume the example referenced in Figure 3. As one would expect the root of the tree covers the full range of keys. At each level the keys are partitioned according to their first differing bit. The first difference occurs between keys k_3 and k_4, thus, the left child of the root node contains keys k_0 to k_3 and the right child contains k_4 to k_7. We continue this process to essentially get a hierarchical representation of the common prefixes between each key. At the bottom level of the tree we will find that each child references a key.

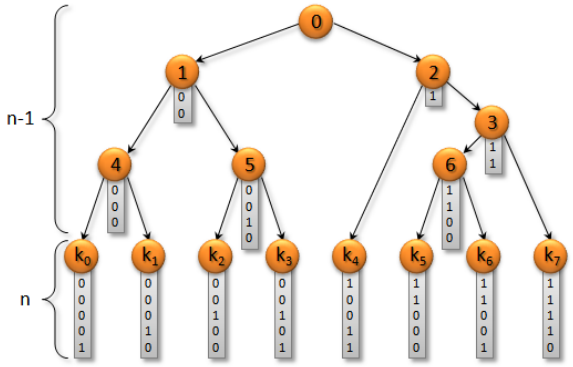


Figure 3: A visual representation of an ordered radix tree. The numbers 0-7 act as keys (leaf nodes) of the tree. Each internal node represents a common range prefix of the binary value of all the leaf nodes bellow it. Notice we have $N-1$ internal nodes for N keys.

A radix tree is considered a compact data structure, as it omits nodes which only have one child, thus removing redundant information and decreasing the overall size of the tree in memory.

One property of a binary radix tree is that any given tree with N keys will have $n-1$ internal nodes. This allows us to know, even before we construct the tree, how many nodes, and thus how much memory, we will require. Assuming that we have a lexicographically ordered tree allows us to express $[i, j]$ as the range of keys covered by any given internal node. We use $\delta(i, j)$ to denote the length of the longest common prefix between the keys k_i and k_j. The ordering of the keys automatically implies that $\delta(i', j') \geq \delta(i, j')$ for any $i', j' \in [i, j]$ [Kar12]. We can then determine the common prefix shared between a key under a given node by comparing the first and last key it covers - all the other keys in between are guaranteed to share the same or a larger prefix.

In practice each internal node partitions the keys under it on their first differing bit, the one following $\delta(i, j)$. We can safely assume that in the range $[k_i, k_j]$ part of the keys will have said bit set to 0 and others will have it set to 1. Since we are working with an ordered tree all of the keys with the bit set to 0 will be presented before the keys with the bit set to 1. We call the position of the last key, where this bit has value 0, the split position denoted by $\gamma \in [i, j-1]$. Since the split position is where the first bit differs between the keys in the range we can say that

$\delta(\gamma,\gamma+1) = \delta(i,j)$. The ranges $[k_i,k_\gamma]$ and $[k_{\gamma+1},k_j]$ will be subdivided at the next differing bit. We can thus say for sure is that $\delta(i,\gamma) > \delta(i,j)$ and $\delta(\gamma+1,j) > \delta(i,j)$. Taking Figure 3 once more as reference, we can see that the first differing bit happens between keys k_3 and k_4, the range is then split at $\gamma = 3$ resulting in the subranges $[k_0,k_3]$ and $[k_4,k_7]$. The left child then splits its range $[k_0,k_3]$ at the third bit, $\gamma = 1$. The right child splits the range $[k_4,k_7]$ at $\gamma = 4$, at the second bit, and so on.

4.2 Morton Codes

We start our process with a series of primitives represented by 3D points. In order to generate our BVH we first start by deciding in which order each leaf node will be represented in the tree. A good approach is to sort the leafs according to their position, generally we will want primitives close to each other in 3D space to appear close to each other on the tree. In order to do this we sort them via a space-filling curve, more specifically a Z-order curve. For this we take the centroid of each triangle and express it relative to the bounding volume of the entire scene, known after loading the scene's data.

Let bvh_{min} be defined as the minimum extents of the scene's bounding volume and bvh_{max} as its maximum. If we define c as the centroid point of a given triangle then we can express q, the same point, but now in coordinates relative to the bounding volume of the scene through the following formula:

$$q = \frac{c - bvh_{min}}{bvh_{min} - bvh_{max}} \tag{1}$$

q's coordinate values now vary between 0 and 1. We can think of it as a point within the 3D space delimited by the scene's bounding volume. The closer each coordinate is to 0 the closer the point will be to the minimum point of the bounding volume, and, the closer it is to 1 the closer it will be to its maximum point.

We now want to express this 3D point as a *Morton code*. The first step in this process is to transform our point from a continuous space into a discrete one. We achieve this by quantizing each floating point coordinate into a range created by the difference between the scene's bounding volume maximum and minimum extremes. Morton codes are most efficiently expressed as a single integer so to represent a 3D point in a single integer value we will have to make some precision sacrifices. We assume a machine architecture with 32 bit sized integers, meaning that in order to represent 3 distinct values in 32 bits we will have 10 bits for each of the 3 Cartesian coordinates. We are thus left with the following quantization equation:

$$q = \frac{(c - bvh_{min}) \times 2^{10}}{bvh_{min} - bvh_{max}} \tag{2}$$

Where q is now a 3D point whose coordinates are composed of 10 bit integers. To correctly represent this point along a Z-order curve we interleave the bits of all three coordinates together to form a single binary number. We take the value of each coordinate, expand the bits by inserting 2 bit "gaps" after each bit and then interleave them. Beyond this point it is simply a matter of calculating the Morton codes for each primitive and sorting them via a parallel sorting algorithm, we used radix sort for this operation.

4.3 Parallel Construction of BVH

This section follows the same line of thought described by Karras in [Kar12]. The idea is to assign indexes for the internal nodes in a way that enables finding their children without depending on earlier results, this way we can fully parallelize the construction of the BVH.

We create two separate arrays to store our nodes, **L** for the leaf nodes and **I** for the internal nodes. We define the layout of **I** as having the root node at the index 0, denoted by $\mathbf{I_0}$. The index of each internal node will be defined by its split position. For any given node the left child will be defined in $\mathbf{I}_\gamma$ if it covers more than one key and at $\mathbf{L}_\gamma$ if it doesn't. Similarly the right child will be located either at $\mathbf{I}_{\gamma+1}$ or at $\mathbf{L}_{\gamma+1}$. This layout has an important property, the index of every internal node will either coincide to the first or the last key it covers. Take for example the root node, it covers the entire range of keys $[0,n-1]$ and is located at position $\mathbf{I_0}$. A node covers the range $[i,j]$; its left child will be located at the end of the range $[i,\gamma]$ and its right child located at the beginning of the range $[\gamma+1,j]$.

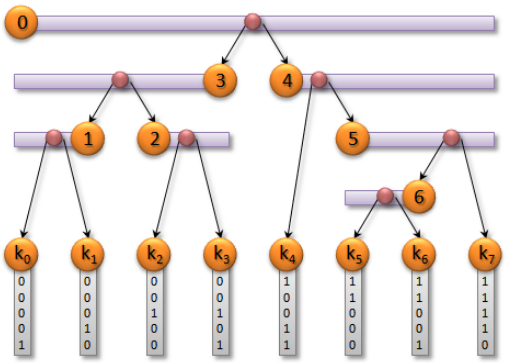


Figure 4: Bar representation of the keys covered by each internal node.

In Figure 4 we present a visual example of this property. Node 0 covers the entire range $[k_0,k_7]$ and is therefore located at $\mathbf{I_0}$. Its children, node 3 and 4 cover the ranges $[k_0,k_3]$ and $[k_4,k_7]$ and are placed at $\mathbf{I_3}$ and $\mathbf{I_4}$, respectively. Interestingly, this process will never result in gaps or duplicates when populating the internal node array. An advantage of using this scheme is that each internal node is conveniently placed next to a sibling

node in memory. And since internal nodes run from 0 to $n-1$ we can use them to directly address the nodes in memory.

In order to construct the BVH, we need to know more than simply which nodes cover which keys, we need to know how the nodes connect amongst themselves, i.e. the parent-child relations.

Lets say we want to determine the direction in which the range of keys covered by node $\mathbf{I_i}$ extends in. We call this direction d. In order to determine d we compare the length of the common prefix between the keys k_{i-1}, k_i, and k_{i+1}. If $\delta(i-1,i) > \delta(i,i+1)$ then $d=-1$. And if $\delta(i-1,i) < \delta(i,i+1)$, then $d=+1$.

Knowing this we can say that k_i and k_{i+d} both belong to node $\mathbf{I_i}$, and that k_{i-d} belongs to node $\mathbf{I_{i-d}}$.

We now need to know how far the range of each node extends. Since k_{i-d} does not belong to the range of keys covered by node $\mathbf{I_i}$, we can safely assume that the keys which are share amongst themselves a larger common prefix than k_i and k_{i-d} do. We call this common prefix δ_{min} so that $\delta_{min} = \delta(i,d-1)$.

$\delta(i,j) > \delta_{min}$ for any k_i belonging to $\mathbf{I_i}$. This being said all we need to do to find the other end of the range is to search for the largest l that satisfies the equation $\delta(i,i+ld) > \delta_{min}$. The fastest way to do this is to start at k_i and in powers of 2 increase the value of l until it no longer satisfies $\delta(i,i+ld) > \delta_{min}$. Once this happens we know that we have gone too far and we have left the range of keys covered by $\mathbf{I_i}$. Lets call this upper bound l_{max}. We know for sure the correct value of l is somewhere in the range $[l_{max}/2, l_{max}-1]$. Now it is only a matter of using a binary search to find the value of l under which $\delta(i,d(l+1)+i) \leq \delta_{min}$.

After finding the value of l we can use $j = i+ld$ to specify the other end of the range.

Lets call δ_{node} the length of the common prefix shared between k_i and k_j, given by $\delta(i,j)$. We use δ_{node} to search the split position γ that partitions the keys covered by $\mathbf{I_i}$. Now we perform a search for the largest $s \in [0, l-1]$ that satisfies $\delta(i,i+sd) > \delta_{node}$. i.e., we need to find the furthest key which shares a larger prefix with k_i than k_j does.

Discovering γ allows us to determine the ranges covered by each children. The left child will have a range covering $[min(i,j]), \gamma]$ and the right child will cover $[\gamma+1, max(i,j)]$.

For the final step we analyze the values of i, j and γ. If $i = \gamma$ we know $\mathbf{I_i}$'s left child is the leaf node $\mathbf{L}_\gamma$, otherwise it's the internal node $\mathbf{I}_\gamma$. Correspondingly if $j = \gamma+1$ we say $\mathbf{I_i}$'s right child is the leaf node $\mathbf{L_{\gamma+1}}$, otherwise it's internal node $\mathbf{I_{\gamma+1}}$. Pseudocode for this algorithm can be found in Algorithm 1.

Algorithm 1 Pseudocode for the parallel construction of a BVH [Kar12].

```
for all internal node with index i ∈ [0, n−2] do
    // Determine direction of the range (+1 or -1)
    d ← sign(δ(i,i+1) − δ(i,i−1))
    // Compute upper bound for the length of the range
    δ_min ← δ(i,i−d)
    l_max ← 2
    while δ(i,i+l_max·d) > δ_min do
        l_max ← l_max·2
    // Find the other end using binary search
    l ← 0
    for t ← l_max/2, l_max/4, ..., 1 do
        if δ(i,i+(l+t)·d) > δ_min then
            l ← l+t
    j ← i+l·d
    // Find the split position using binary search
    δ_node ← δ(i,j)
    s ← 0
    for t ← {⌈l/2⌉, ⌈l/4⌉, ..., 1} do
        if δ(i,i+(s+t)·d) > δ_node then
            s ← s+t
    γ ← i+s·d
    // Output child pointers
    if min(i,j) = γ then left ← L_γ else left ← I_γ
    if max(i,j) = γ+1 then right ← L_{γ+1} else right ← I_{γ+1}
    Ii ← (left, right)
```

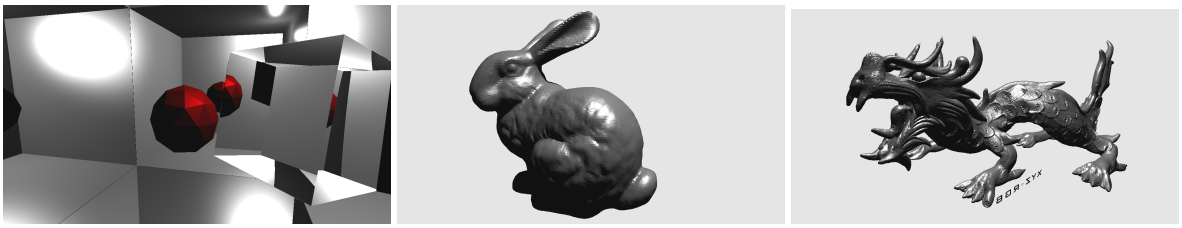
4.4 Compression

Reducing memory footprint and bandwidth is our goal. A certain amount of bandwidth is reduced by simply using a BVH. We can further improve our gains by compressing both the BVH and the triangle mesh.

4.4.1 BVH Compression

We start a thread at each internal node and make our way up to the top of the tree. Once we reach the top of the tree we start walking the same path in the opposite direction, that is, from the root node to the internal node in question. As we descend each level we take the bounding box of the previous parent and subdivide each dimension in 1024 segments. We treat this 1024^3 grid as a voxel space and map the minimum and maximum points of the current level's node bounding box into the nearest corresponding voxels.

It becomes clear that we lose some precision as we descend each level since we are going from floating point coordinates into integer indexes. Each index will be contained in the range $[0, 1024[$, so we can store 3 of these indexes in a single 32 bit integer. Each bounding box can then be stored as a single 2 integer data structure instead of a 6 float data structure, reducing its size down from 24 bytes to 8 bytes. It is obvious that with this method our BVH will become more loosely coupled since we lose precision when going from a continuous space (world coordinates) into a discrete one (voxel indexes). We round up when mapping the maximum point of a bounding volume to a voxel and round down when mapping the minimum so our bounding



(a) SHINY (152 triangles) (b) BUNNY (70K triangles) (c) ASIAN DRAGON (7.2M triangles)

Figure 5: Test Scenes.

volumes remain coherent. We predict this loss of precision will not have a big impact in our intersection tests, we will most likely intersect a few more bounding volumes, as they will be slightly larger in size, but we do not expect a significant increase in the number of tests. One thing to take into account is that in each thread we must quantize the entire path from the root to each internal node otherwise our BVH will become incoherent.

4.4.2 Mesh Compression

Karras [Kar12] uses each leaf node as a redirect to a primitive. We can remove an indirection level by having each leaf node envelop a single triangle and store said triangle in the node itself. We can also try and diminish the number of memory necessary to store a triangle by following the same line of thought from the BVH compression. By quantizing the position of each vertex of each triangle, into the bounding box that encapsulates it, we can go from having to store 9 floats to just 3 integers. This will of course have an impact on the model itself as we will be losing the original coordinates of each vertex. When recalculating each vertex back to world coordinates we will be changing the final world positions of each triangle, resulting in slight mesh deformation. Interestingly enough, this effect isn't noticeable unless examining models up close. In a practical application like a game or simulation where the camera and the objects are constantly moving this effect could pass up unnoticed. This compression step can be done in the same kernel as the quantization of the internal nodes of the tree, thus having little to no effect on the post-processing time of the BVH.

5 RESULTS

Tests were made using an NVIDIA GeForce GTX 970 with 4 GB of RAM. We chose to render our images in 1280x720p as this is one of the most commonly used resolutions for multimedia content.

We tested our algorithm with three different scenes, ASIAN DRAGON, SHINY and BUNNY.

SHINY (see Figure 5a) is a scene representative of highly reflective and refractive scenes. It also represents a low polygon scene. It consists of an icosphere surrounded by five mirrors and a glass prism. This scene focuses on testing performance in scenes with a high number of secondary rays.

BUNNY (see Figure 5b) is what we would call a "medium" sized scene. It serves as a midterm between low polygon scenes (SHINY) and high polygon scenes (ASIAN DRAGON). This scene only features eye and shadow rays.

ASIAN DRAGON (see Figure 5c) is representative of complex objects with a high number of triangles. It is a dense model with a great number of triangles in a small space. Our intent with this scene is measure the performance gains of BVH compression for dense BVHs. This scene only features eye and shadow rays.

5.1 Ray-Box Intersection Tests

Despite SHINY being a small, enclosed scene we notice that lack of precision of the bounding volume areas causes an increase of 9% in the intersection tests (see Figure 6). We assume this is caused by reflected and refracted rays, that would normally pass tangent to the icosphere, but are instead tested and categorized as a hit. In BUNNY we notice an increase of 7.1% in the number of ray-box intersection tests, making it the less affected of all the 3 scenes. In ASIAN DRAGON we notice an increase of 12.38% in the number of ray-box intersection tests. Since this scene has a high tree we expect rounding "errors" to accumulate over the levels thus leading to this increase in the number of intersection tests.

5.2 Ray-Triangle Intersection Tests

SHINY has an increase of 11.1% in the tests performed. We believe this number is greater, in comparison to the other test scenes, because in this scene almost every ray will hit an object and thus it will have to traverse the entire BVH, making it more prone to the effects of loss of precision of the bounding volumes.

In BUNNY we notice an increase of 7.4% in the number of tests performed. This increase in the number of ray-triangle test seems to be similar to the increase of ray-box intersection tests.

ASIAN DRAGON has an increase of 10.75% tests performed. As in BUNNY this number seems to go in hand

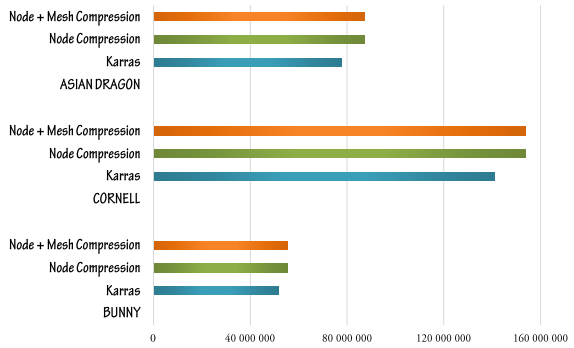


Figure 6: Amount of ray-box intersections.

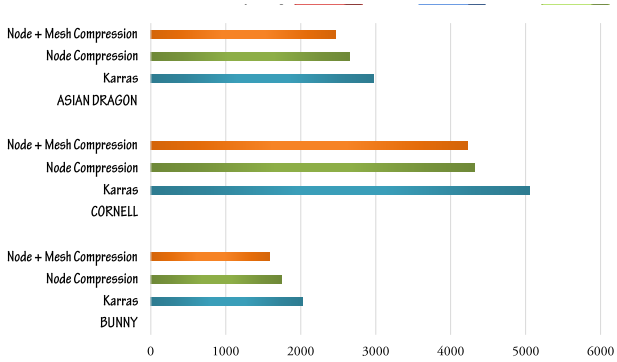


Figure 7: Memory read in each frame (MB).

with the increase of ray-box intersection tests, although slightly lower.

5.3 Global Memory Reads

The following results are based in the number of intersection tests performed and the size of each internal node, leaf node and primitive.

In SHINY we notice we are able to achieve a significant impact in the amount of memory read with node compression (14.5%), but see little to no gains when implementing mesh compression (see Figure 7). This is because, in this scene, the number of ray-box intersections dwarfs the number of ray-triangle intersections, hence optimizations made to the execution time of ray-triangle tests will have little impact.

BUNNY reduces its memory accesses by 14% when compressing internal nodes alone and 22% when compressing the mesh alike. These results go in hand with the results obtained in ASIAN DRAGON providing some insight that every reasonably complex model is positively affected by our improvements.

In ASIAN DRAGON we see a steady decrease of accessed memory with our algorithms. We reduce memory accesses by 11.9% with node compression and, by 20.3% with mesh compression.

5.4 Kernel Execution Time

In this section we measure the execution time for each of the most relevant kernels. KARRAS represents the basic algorithm, as described by Karras in [Kar12]. NODE is the same algorithm with BVH compression. NODE+MESH is the same algorithm with BVH and triangle compression.

SHINY does not benefit significantly from our modifications. We notice a reduction in rendering time of 6.7% when compressing internal nodes and 10.7% when compressing the mesh. BVH construction and compression time improvements are marginal at best. We believe our modifications do not have an impact on this scene since it's easy to fit in the GPU caches.

	KARRAS TIME (MS)	NODE TIME (MS)	NODE+MESH TIME (MS)
BVH CREATION	2.17	2.17	2.17
NODE COMP	0	0.01	0.01
MESH COMP	0	0	0.01
RENDERING	48.93	45.68	**43.68**

Table 1: Kernel execution time for SHINY frame.

	KARRAS TIME (MS)	NODE TIME (MS)	NODE+MESH TIME (MS)
BVH CREATION	7.82	7.82	7.82
NODE COMP	0	0.24	0.24
MESH COMP	0	0	0.35
RENDERING	80.74	64.15	**46.83**

Table 2: Kernel execution time for BUNNY frame.

In the BUNNY scene we notice most of the time is spent executing the rendering kernel as in SHINY. The creation of the BVH has a slight impact on the time required to render each frame and remains constant across all 3 variations. Compression is quick but greatly impacts the time it takes to render the scene. Here we can see a reduction in rendering time of 20.55% when compressing the BVH's internal nodes and a reduction of 41.01% when compressing both internal nodes and triangles. We are able to achieve a 37.62% overall improvement in time to frame performance with a full rebuild.

	KARRAS TIME (MS)	NODE TIME (MS)	NODE+MESH TIME (MS)
BVH CREATION	424.64	424.64	424.64
NODE COMP	0	16.33	16.33
MESH COMP	0	0	24.43
RENDERING	246.36	194.44	**160.95**

Table 3: Kernel execution time for ASIAN DRAGON frame.

In ASIAN DRAGON we see a time reduction of 21.08% in rendering kernel execution when compressing just the internal nodes, and 34.67% when compressing both internal nodes and triangles. As we can see our algorithm has a significant impact in high poly models, we fetch a large number of BVH nodes in this scene. Un-

fortunately the BVH CREATION kernel takes up most of the time in each frame when dealing with such a high number of polygons. So overall we are only able to achieve a 6.6% improvement in time to frame performance with a full rebuild.

6 CONCLUSIONS

In our tests we used the vanilla version of Karra's algorithm as a control benchmark. Tests showed all complex test scenes benefit from our compression approach. As we hypothesized the number of intersection tests increases but the overall time to traverse the BVH and perform each intersection test decreases. Some scenes benefit more from our algorithm than others. We notice that in scenes with a high number of polygons the construction of the BVH becomes a bottleneck, taking most of the time in the frame. This affects our algorithm as much as the control algorithm. Gains from our approach were diluted by this bottleneck for this kind of scene.

As we initially expected real time ray tracing benefits from a reduction in memory footprint and bandwidth. Despite having to spend more time compressing and then decompressing both the BVH and the scene's primitives in each frame this penalty is compensated by the reduced time it takes to fetch the scene's data from global memory.

7 FUTURE WORK

BVH construction times are a bottleneck in high polygon scenes. Our compression techniques improve rendering times, but the construction times are still an issue, in particular in large scenes with full rebuilds, these could also benefit from working set minimization, compression, or both. Using 64 bit types to store quantized values could also be a solution worth exploring.

8 ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their insightful comments.

This work was supported by national funds through Fundação para a Ciência e Tecnologia (FCT) with reference UID/CEC/50021/2013.

REFERENCES

[BEM10] Pablo Bauszat, Martin Eisemann, and Marcus A Magnor. The Minimal Bounding Volume Hierarchy. In *VMV*, pages 227–234, 2010.

[GPKB12] Jayshree Ghorpade, Jitendra Parande, Madhura Kulkarni, and Amit Bawaskar. GPGPU processing in CUDA architecture. *arXiv preprint arXiv:1202.4347*, 2012.

[GPM11] Kirill Garanzha, Jacopo Pantaleoni, and David McAllister. Simpler and faster HLBVH with work queues. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, pages 59–64. ACM, 2011.

[KA13] Tero Karras and Timo Aila. Fast parallel construction of high-quality bounding volume hierarchies. In *Proceedings of the 5th High-Performance Graphics Conference*, pages 89–99. ACM, 2013.

[Kar12] Tero Karras. Maximizing parallelism in the construction of BVHs, octrees, and k-d trees. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*, pages 33–37. Eurographics Association, 2012.

[LGS+09] Christian Lauterbach, Michael Garland, Shubhabrata Sengupta, David Luebke, and Dinesh Manocha. Fast BVH construction on GPUs. In *Computer Graphics Forum*, volume 28, pages 375–384. Wiley Online Library, 2009.

[Mor66] Guy M Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company New York, 1966.

[MW06] Jeffrey Mahovsky and Brian Wyvill. Memory-Conserving Bounding Volume Hierarchies with Coherent Raytracing. In *Computer Graphics Forum*, volume 25, pages 173–182. Wiley Online Library, 2006.

[PL10] Jacopo Pantaleoni and David Luebke. HLBVH: hierarchical LBVH construction for real-time ray tracing of dynamic geometry. In *Proceedings of the Conference on High Performance Graphics*, pages 87–95. Eurographics Association, 2010.

[Whi80] Turner Whitted. An Improved Illumination Model for Shaded Display. *Commun. ACM*, 23(6):343–349, Jun 1980.

[WMG+09] Ingo Wald, William R Mark, Johannes Günther, Solomon Boulos, Thiago Ize, Warren Hunt, Steven G Parker, and Peter Shirley. State of the art in ray tracing animated scenes. In *Computer Graphics Forum*, volume 28, pages 1691–1722. Wiley Online Library, 2009.

Real-time 3D Gesture Recognition using Dynamic Time Warping and Simplification Methods

Alan dos Santos Soares
Federal University of Bahia, Brazil
Av. Adhemar de Barros, Ondina
40.170-110, Salvador, Bahia
alan.soares@ufba.br

Antonio L. Apolinário Jr
Federal University of Bahia, Brazil
Av. Adhemar de Barros, Ondina
40.170-110, Salvador, Bahia
antonio.apolinario@ufba.br

ABSTRACT

The recognition of dynamic gestures of hands using pure geometric 3D data in real-time is a challenge. RGB-D sensors simplified this task, giving an easy way to acquire 3D points and track them using the depth maps information. But use this collection of raw 3D points as a gesture representation in a classification process is prone to mismatches, since gestures of different people can vary in scale, location and velocity. In this paper we analyze how different techniques of simplification and regularization can provide more accurate representations of the gestures. Using Dynamic Time Warping (DTW) as the classification method, we show that the simplification and regularization steps can improve the recognition rate and also reduce the time of gesture recognition.

Keywords

Hands Gesture; Geometric Modeling; 3D Gesture Classification; Real-Time; Gesture Acquisition;

1 INTRODUCTION

The use of hand gestures in the construction of computer systems has many challenges [Pal+13]. For example, a single gesture can have different meanings depending on the culture of each country or region [HK12]. Furthermore gestures of different people can vary in scale, location and velocity.

The complexity of a gesture depends on the amount of body parts used in the movement [Pal+13], so it is necessary to define a descriptor or method to simplify the gesture in such a way that only key points are stored to improve the performance of the recognition. Our work provides an approach that recognizes gestures in real-time, regardless of position, lighting and physical aspects of the user. We define gesture as sequence of hand positions performed in the 3D space like "let's go", "bye bye", etc. We evaluated the recognition rate and performance using different combinations of regularization and simplification methods.

The main contributions of this paper are:

- A purely geometrical approach to gesture recognition.
- A method to recognize gestures in sequence without human intervention, requiring only an time interval between the executions.
- A comparison of different methods for curve simplification, showing that a step of pre-processing can reduce about a half the time consumption needed to recognize gestures.
- Support to recognize gestures with one or both hands, trained or not.

As an secondary contribution, we create a new dataset composed of depth, image and tracking information for 7 gestures performed by 7 people with different physical aspects, totalizing a set of 1099 executions.

This article is structured as follows: in Section 2 we discuss the advantages and drawbacks of related works. In Section 3 we show our proposed solution in detail. In Section 4 we present the results of the evaluation of different simplification and regularization methods through the performance and recognition aspects. In Section 5 we state our final remarks and suggest some future works.

2 RELATED WORK

Many works for gesture recognition have been published in the last few years [RA15] [Che+13] [HK12] [MA07].

The gesture recognition approaches can be divided in three main: glove-based, vision-based and depth-based. The glove-based approach uses a device to capture the 3D information (position and orientation) about hands or fingers directly, having the advantage of less input data and high speed [Bar+15]. However, the device has to be used all the time, besides it has a lot of cables and is considered more invasive [HK12]. Vision-based approaches are less invasive that the glove-based ones because the user does not need to use wearable devices [RA15]. However, vision-based approaches have

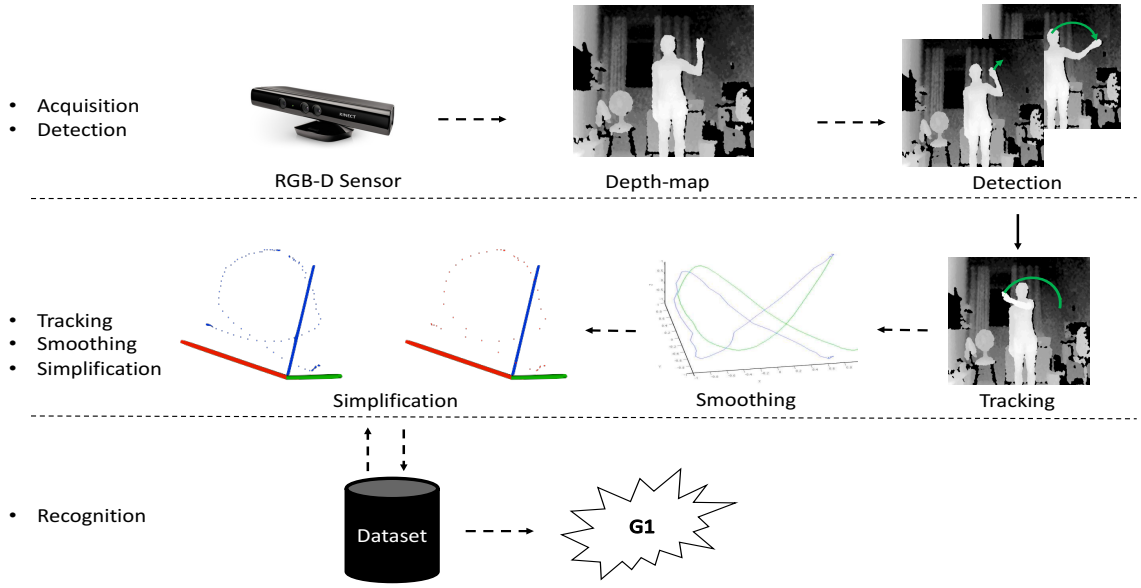


Figure 1: Overview of the proposed approach.

some disavantages like sensitiveness to lighting, color and shadow, limited acquisition by the distance, and the 3D information cannot be obtained directly [Han+13] [HK12].

Recent works uses the Dynamic Time Warping (DTW) [RK05] algorithm to recognize gestures. The DTW finds the cost of similarity through the alignment of two time series. [Iba+14] proposed a framework called EasyGR (Easy Gesture Recognition) that assists developers in the implementation of NUI applications, reducing the complexity through the encapsulation of the algorithms and management of the gesture data. [Bau+13] improved the recognition rates compared to the usual DTW [SC07] and Hidden Markov Model (HMM) [Rab89] using a probability-based DTW that updates the cost of the DTW according to a Gaussian model [Mat01]. However, both approaches were not evaluated taking into account the gestures obtained by people with different physical aspects.

[Wu+14] proposed a new method for view-invariant gesture recognition using a shape representation that is build from a set of euclidean distances between all trajectory points. The shape is smoothed using a ten-order B-Spline interpolation and the classification was performed using a Support Vector Machine (SVM) classifier [Mul+01]. Other approach [Bar+15] uses a wearable camera coupled to the user's head to recognize gestures performed by hands. This type of approach is can generate movement restriction, as it relies on wearable devices, batteries, cables, besides it can only recognize gestures that are in the field of view of the camera.

Our work provides an approach that use purely geometric information to recognize gestures. Different of some related works [Wu+14], we do not need train a model to recognize gestures. Different of [Bar+15], as we use a RGB-D sensor users do not need to use wearable devices to perform gestures. We introduce a step of simplification that can reduce the time consumption to recognize a gesture. This step allow the recognition in real-time.

3 PROPOSED SOLUTION

Our solution aims to use simplification and regularization techniques to speedup the recognition in real-time. We proposed an approach that is focused only in the use of geometric information.

The figure 1 shows an overview of our approach. First we use a RGB-D sensor (Microsoft Kinect) to capture the geometric information through the depth-map. Then, we use an algorithm [Sen13] to detect and track the 3D hands movement. The next step smooths and simplifies the gesture to remove noise and capture the key points. Finally, we use the DTW to classify the gesture.

3.1 Detection and Tracking

The first step aims to detect the hands using an RGB-D sensor. The RGB-D sensor used to acquire the depth information was the Kinect [CLV12]. We use a sample algorithm of the OpenNI [1] library to detect and track the movement of the hands. Detection begins from the execution of some of the basic gestures implemented by OpenNI, such as "bye bye". After the hand detection, the algorithm is able to track the movement of the hands, providing the central position $P_i(x,y,z)$ of the hand in each frame.

[1] http://www.openni.org/

We have developed an algorithm to automatically detect when a gesture starts and ends. This allow the continuous gesture recognition without human intervention using only a time interval between gestures. The gesture start consists of verifying if the sum S of the distances D_i between the previous n positions is greater than a threshold t.

$$S = \sum_{i=1}^{n-s} D_i \tag{1}$$

If the sum S is greater than the threshold t, then the hand is in motion, otherwise it is stopped. We choose $n = 25$ 25 empirically to detect the hand's movement.

3.2 Normalization

The next step normalizes the gesture since it can vary in scale and location. We use the same normalization proposed by [Iba+14] first calculating the centroid c_i of the gesture by dividing the sum of the points by the total number of points n of the gesture.

$$c_i = (\bar{x}, \bar{y}, \bar{z}) = \frac{\sum_{i=1}^{n}(x_i, y_i, z_i)}{n} \tag{2}$$

Then, the centroid is used to move all points to the origin with (3), that subtracts from each point of the gesture the respective coordinate of the centroid.

$$(x_i, y_i, z_i)' = (x_i - \bar{x}, y_i - \bar{y}, z_i - \bar{z}) \tag{3}$$

In the end, we scale the gesture in the interval -1 and 1. These processes ensures that the gesture is recognized regardless of the location and physical aspects of the user.

3.3 Smoothing

Once normalized the gesture, the next step smooths the raw gesture data to reduce noise by the depth sensor. The method used was the Laplacian [Tau95], which consist of recalculating all points using the mean of each point and its neighbors, according to (4). In our approach we used the 1-ring neighbourhood to smooth one time the gesture. Figure 2 shows the smoothing.

$$\bar{\mathbf{x}_\mathbf{i}} = \sum_{i=1}^{n-1}\left(\frac{x_i}{x_{i-1} + x_i + x_{i+1}}\right) \tag{4}$$

3.4 Simplification

After the normalization and smoothing of the gesture, the next step consists on its simplification. We use this step to provide a compact representation of the gesture, further improving the performance of the gesture recognition.

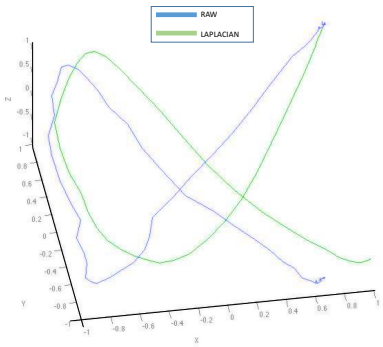


Figure 2: A raw gesture before and after smoothing using the Laplacian operator.

We use two algorithms to perform this task. The first approach simplifies the gesture using a curvature-based method and the second uses the Douglas-Peucker (DP) algorithm [DP73]. As we will show in the section 4, both of them keep the high recognition rate, while improving the algorithm's performance.

3.4.1 *Curvature*

The first simplification method of the gesture consists in checking whether the curvature of each segment is below a pre-defined threshold $t = 0.01$. In section 4.2 we explain how we choose this value.

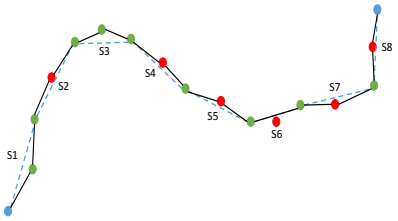


Figure 3: The endpoints in blue cannot be removed. The green points are points evaluated that cannot be removed. The red points can be removed because its curvature are below the threshold t.

As shown in Figure 3, the curvature-based method evaluates the curvature iteratively using segments defined by a point and its neighborhoods, e.g. $S_i = (p_{i-1}, p_i, p_{i+1})$. Then, for each segment S_i, if the curvature of S_i is below the threshold t, then the middle point p_i is removed. The point p_{i-1} of the next segment is the point p_{i+1} of the previous segment.

3.4.2 *Douglas-Peucker*

The Douglas-Peucker (DP) algorithm introduced by [DP73] is a polyline simplification. As shown in the

figure 4, the DP algorithm first use the endpoints $[A,B]$ to find and calculate the distance to the furthest point C. Then it uses the points $[A,C]$ and $[B,C]$ to calculate again the furthest points of $[A,C]$ and $[B,C]$, that is D and E. Finally it add the points $[C,D,E]$ if distance exceeded the tolerance t. This condition of similarity is based on the maximum distance measured between the original and the simplified curve. The original endpoints always are inserted in the simplified curve.

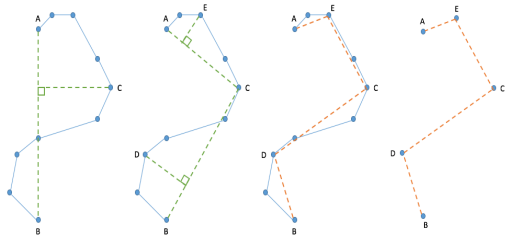


Figure 4: Steps of the DP algorithm to simplify a curve with 10 points.

We choosed this algorithm because it can reduce the number of points of the gesture while retaining its shape. Furthermore, the DP algorithm is faster then others algorithms like Bend Simplify [VH99].

3.5 Recognition

The last step use Dynamic Time Warping (DTW) [RK05] as classification method to recognize gestures. We use the nearest neighborhood algorithm with DTW to find the closest gesture according with the cost distance provided by it.

3.5.1 Dynamic Time Warping - DTW

The DTW algorithm finds the cost of similarity through the alignment of two time series, which in our case are the gestures. The basic idea is to construct an array of distances between the two trajectories and find the minimum distance between each pair of points. The result of the comparison is the sum s of the smallest distances found. The lower the value of s, the higher the degree of similarity between the two trajectories.

One of the main advantages of using the DTW is that it allows to compare two trajectories, even if they have different lengths [RK05]. This property of the DTW is important, since the gestures can be done with different speeds, so that the sampling rate is not always the same.

4 EXPERIMENTAL RESULTS

This section describes in detail the experimental setup and results. First we did a cross-validation to evaluate different parameters and obtain the optimal values to use in the Curvature, DP and DTW algorithms. Then we use the parameters found to evaluate the recognition rate and performance for each class of gesture applying these algorithms. Also, we evaluate the recognition and performance using as template the median gesture from each class.

The experimental evaluation was performed in a Macbook Pro (13-inch, Late 2011), Processor 2.4GHz Intel Core i5, Memory 4GB 1333 MHz DDR3, Intel HD Graphics 3000 384 MB, macOS Sierra version 10.12.3.

4.1 Dataset

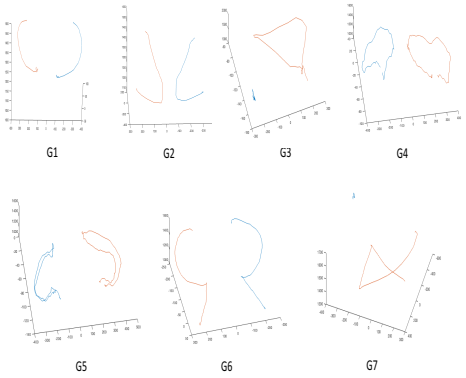


Figure 5: Set of 7 gestures of our dataset.

We tried use the MSRC-12 [Fot+12] dataset, but it has gestures written continuously in a single file and we did not have the executions separator to do it. ChaLearn [Guy+14] has only RGB and depth videos. Therefore, we create our own dataset.

Our dataset contains 1099 gestures, collected from 7 individuals performing 7 gestures, with different physical aspects and positions. In our dataset, we have 5 gestures performed using two hands and 2 gestures performed by one hand. The gestures were recorded using the Kinect XBox 360 sensor at a sample rate of 30Hz. We recorded both RGB, depth and 3D motion of hands, but we only used the motion of hands. Each motion contains a set of 3D positions of both hands. Figure 5 shows our 7 gestures, 6 of them are the same defined in dataset [Fot+12].

After create the dataset, we splits our dataset with 1099 gestures to perform the evaluation using 70% for test and 30% as template matching for each class of gesture. The next step was to generate 7 median gestures from our dataset to use as template and evaluate the recognition rate using all 1099 as tests.

To generate median gestures, we first apply for each gesture class a method to normalize the distance between points according with the desired Euclidean dis-

tance $k = 0.1$. The method calculate the Euclidean distance d_i of each segment s_i and remove the point p_{i+1} if $d_i < k$, otherwise we apply a linear interpolation in the segment s_i until $d_i < k$. Then, we equalize the number of points removing or adding points according with the average point of each gesture class. Finally, we calculate a simple mean for each gesture class g_i with (5), where we divide the sum of x_i, y_i and z_i by the total number of gestures n of each class.

$$g_i = \frac{\sum_{i=1}^{n}(x_i, y_i, z_i)}{n} \quad (5)$$

4.2 Cross-validation

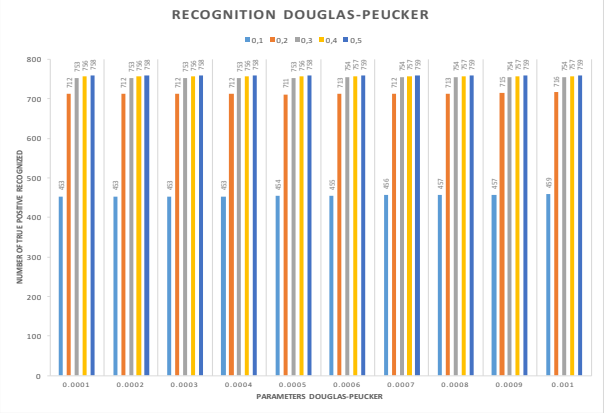


Figure 6: Cross-validation applied for DP with DTW using the parameters in Table 1.

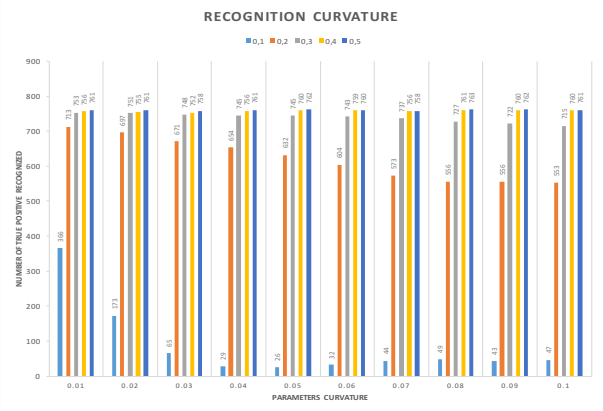


Figure 7: Cross-validation applied for Curvature with DTW using the parameter shown in Table 1.

Algorithm	Parameter domain	Final
Curvature	$0.01 <= t <= 0.1$	0.01
DP	$0.0001 <= t <= 0.001$	0.001
DTW	$0.1 <= d <= 0.5$	0.5

Table 1: Table with the cross-validation parameter domains for each algorithm and the chosen ones.

To perform cross-validation, we use all the gestures in our dataset, totaling 1099 gestures of different classes. We selected 30% of each class for tests, being the same applied for the different combinations of parameters.

After the data preparation phase, we selected a parameter domain for each method described in Table 1. The domains of the parameters were defined according to the normalization of the gesture in the interval of -1 to 1 and in some values tested manually to find the minimum and maximum thresholds of each algorithm. The best threshold criteria was the recognition rate resulting from each combination. The column Final in the Table 1 shows the selected final parameters for each method.

Figures 6 and 7 show the cross-validation result applied in the DP and Curvature algorithms for each DTW parameter shown in Table 1.

4.3 Recognition rate

We have created an algorithm to automate the execution of the tests. Initially the algorithm loads and divides the data into test and template according with section 4.1. Then, the pre-processing is applied for each test iteration before the classification using the DTW algorithm. We save in a file all the parameters used, including the time needed to process and classify the gesture.

As we can see in the Table 2, the results show a recognition rate above 90% with 83.75% on average, even applying a simplification step. The Laplacian with DP provides an improvement of 1.73% compared to the recognition with raw data simplification. Compared to the DTW results of [Iba+14], our approach with simplification showed an improvement of 2% in the recognition rate. We noticed that some classifications of the $G1$ gesture always made matching with the $G6$ gesture, however the opposite did not occur. The same occur to the one hand gestures $G3$ and $G7$, where the recognition rate for $G3$ was 100% while the $G7$ had an average of 94.5%.

Table 3 show the results using median gestures as template. We get a reduction in the recognition rate for the gesture $G2$, where we identified matching with $G1$ that are similar with it. Using the DP algorithm we get an improvement of $47,62\%$ over the raw data. In general the recognition rate was above 90%.

We also identified during the cross-validation process that the variation of the parameter for simplification using curvature did not affect the recognition rate for an evaluation of ten-order threshold. On the other hand the parameter for the DP reduced the recognition rate for larger values, being not robust to variation.

4.4 Performance

Figures 8 and 9 shows the performance results using simplification and regularization methods. In figure 8, compared to the classification with the raw data, the time needed to recognize the gesture was reduced more

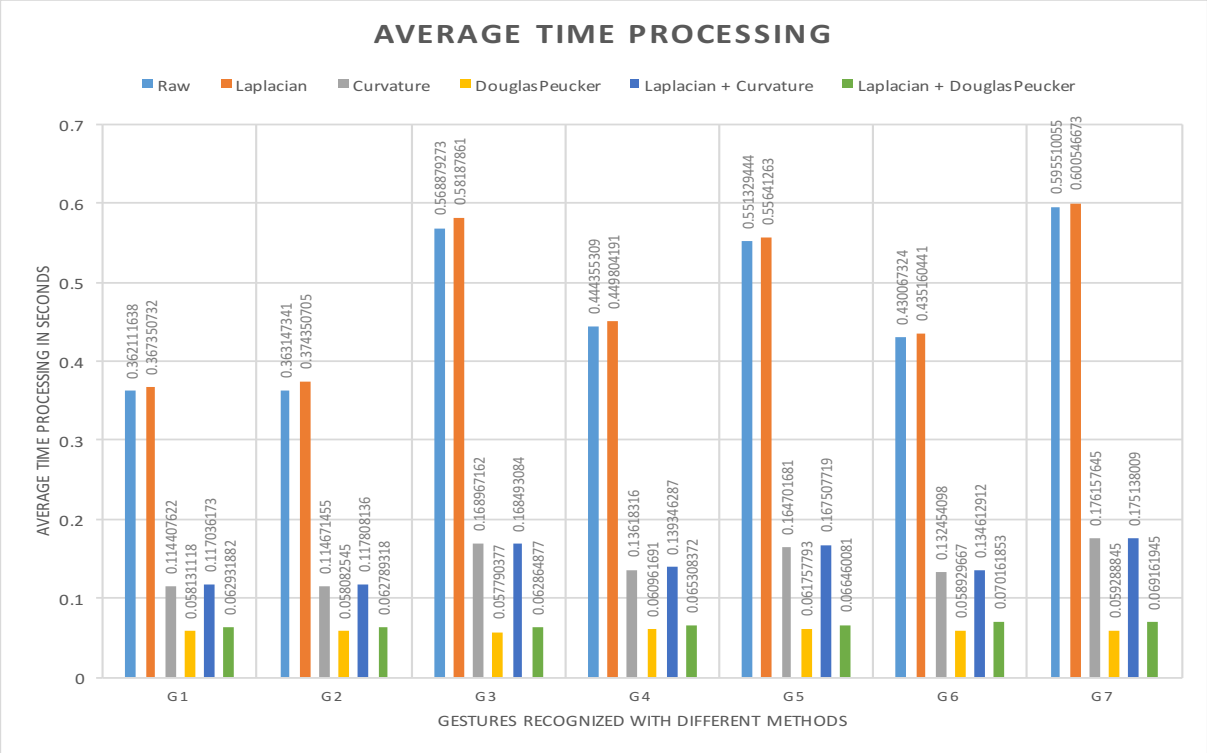


Figure 8: Average time processing for recognize each class of gesture. The process includes both pre-processing and classification time.

	Raw	Laplacian	Curvature	DP	Laplacian + Curvature	Laplacian + DP
G1	90.55	90.55	90.55	90.55	90.55	90.55
G2	100	100	100	100	100	100
G3	100	100	100	100	100	100
G4	98.93	98.93	98.93	98.93	98.93	98.93
G5	100	100	100	100	100	100
G6	100	100	100	100	100	100
G7	94.55	91.81	94.54	95.54	93.63	97.27

Table 2: Recognition rate using different combinations of algorithms for simplification and smoothing for each gesture class.

than a half using a simplification step. Also, figure 9 shows that the average time processing to recognize with median gestures as templates was lower then 2 milliseconds. The Curvature was better then DP for median gestures.

The gestures *G*3, *G*5 and *G*7 had a longer processing time because they are more complex, as can be seen in figure 5. The difference between the raw and Laplacian gestures was very subtle, with a slight increase in the time of recognition with the Laplacian, since it only smooths without simplifying the gesture.

4.5 Discussions

As shown in this section, the recognition rate does not changed significantly when we apply a step to simplify the gestures. However, the performance was reduced more then a half when we apply the simplification. Furthermore, the median gestures reduced the average time processing to 2 milliseconds. With median gestures we allow recognize without compare all samples of the dataset.

The DP algorithm shown better results in performance, but lost for curvature in the recognition rate. We note that the curvature-based method is more robust in simplification in the sense of maintaining the key points that describe the shape of the gesture. This explains why the curvature algorithm obtained better recognition rate results than DP and because DP processing time was better. The DP tends to remove more points in the simplification.

As we conclude, the filter is interesting because can improve the performance without affect the recognition using specifically the DTW.

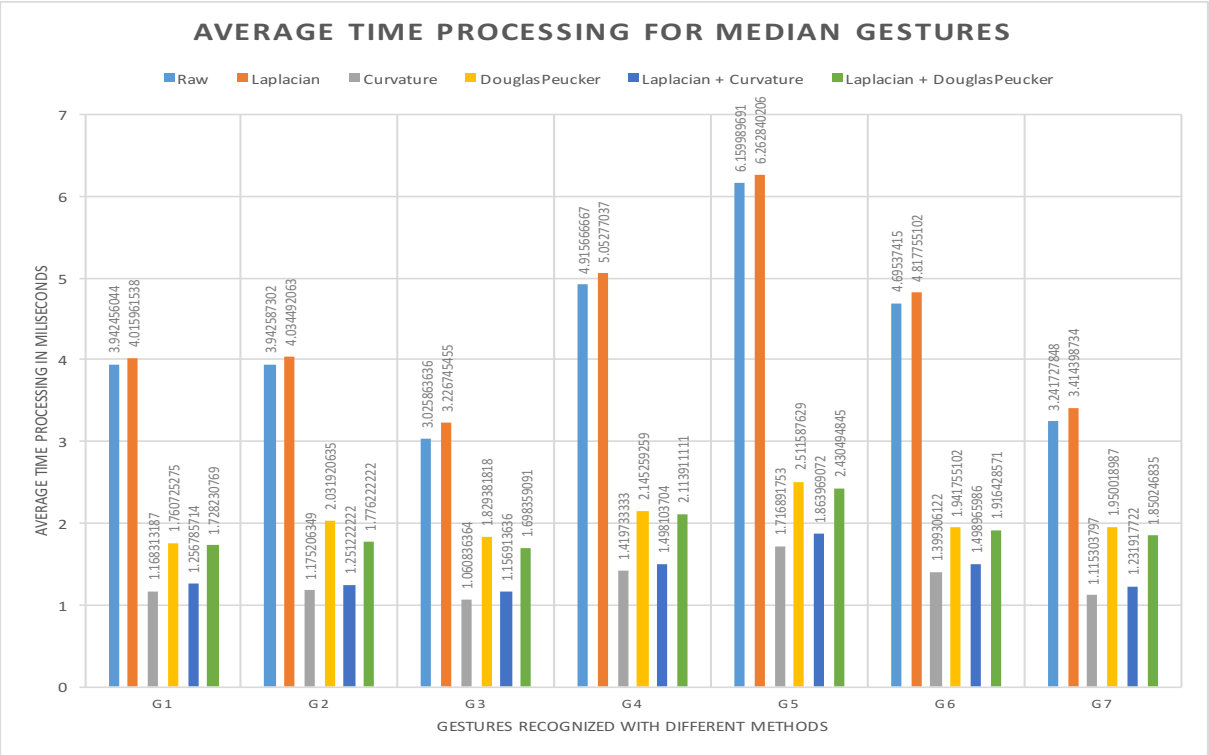


Figure 9: Average time processing for recognize each class using as template the median gesture. The process includes both pre-processing and classification time.

	Raw	Laplacian	Curvature	DP	Laplacian + Curvature	Laplacian + DP
G1	93.40	93.40	93.40	93.95	93.40	93.40
G2	44.44	44.44	47.61	92.06	52.38	79.36
G3	100	100	100	100	100	100
G4	99.25	99.25	99.25	99.25	99.25	99.25
G5	91.23	92.78	93.29	93.29	93.29	93.29
G6	100	100	100	98.63	100	97.27
G7	100	100	100	100	100	100

Table 3: Recognition rate using as template median gestures and different combinations of algorithms for simplification and smoothing.

5 CONCLUSION

In this paper, we propose an approach to gesture recognition based on geometric data and simplification of its representation. We analyzed two simplification methods based on curvature and DP algorithm. In the first, we obtained a recognition rate of 97.7% on average, while for the DP algorithm, we have obtained 98.1%. Using median gestures, we obtained a recognition rate above 90% with exception of the gesture $G2$. Both simplification methods evaluated reduced the recognition time in more than 2 times, being the DP more efficient for the first case, while for median gestures the Curvature was better then DP.

Simplification plays an important role in gesture recognition systems that have large robust datasets. The classification in such systems can not be robust in real-time without a pre-processing step because we noted in our results that performance depends of the number of gestures and points. This makes sense, because we must compare all gestures template to ensure the best match. One of the more important advantages of the simplification is the recognition time reduction.

As future work, we want to create more sophisticated gesture descriptor and use it with a tree decision to avoid full comparison of gestures in the dataset. We also will evaluate the simplification in supervised approaches with HMM to check if the recognition keeps robust after the simplification. As future work, we will also use standard datasets to evaluate our approach. Finally, we will try to recognize gestures continuously, without human intervention and without the need for time intervals between the beginning and end between the gestures.

6 REFERENCES

[DP73] David H Douglas and Thomas K Peucker. "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature". In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10.2 (1973), pp. 112–122.

[Rab89] Lawrence R Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.

[Tau95] Gabriel Taubin. "A signal processing approach to fair surface design". In: *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. 1995, pp. 351–358.

[VH99] Mahes Visvalingam and Simon Herbert. "A computer science perspective on the bendsimplification algorithm". In: *Cartography and Geographic Information Science* 26.4 (1999), pp. 253–270.

[Mat01] Mark W Matsen. "The standard Gaussian model for block copolymer melts". In: *Journal of Physics: Condensed Matter* 14.2 (2001), R21.

[Mul+01] K-R Muller et al. "An introduction to kernel-based learning algorithms". In: *IEEE transactions on neural networks* 12.2 (2001), pp. 181–201.

[RK05] Chotirat Ann Ratanamahatana and Eamonn Keogh. "Three myths about dynamic time warping data mining". In: *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM. 2005, pp. 506–510.

[MA07] Sushmita Mitra and Tinku Acharya. "Gesture recognition: A survey". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.3 (2007), pp. 311–324.

[SC07] Stan Salvador and Philip Chan. "Toward accurate dynamic time warping in linear time and space". In: *Intelligent Data Analysis* 11.5 (2007), pp. 561–580.

[CLV12] Leandro Cruz, Djalma Lucio, and Luiz Velho. "Kinect and rgbd images: Challenges and applications". In: *Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2012 25th SIBGRAPI Conference on*. IEEE. 2012, pp. 36–49.

[Fot+12] Simon Fothergill et al. "Instructing people for training gestural interactive systems". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2012, pp. 1737–1746.

[HK12] Haitham Sabah Hasan and S. Abdul Kareem. "Human Computer Interaction for Vision Based Hand Gesture Recognition: A Survey". In: *2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)* (Nov. 2012), pp. 55–60.

[Bau+13] Miguel Angel Bautista et al. "Probability-based dynamic time warping for gesture recognition on RGB-D data". In: *Advances in Depth Image Analysis and Applications*. Springer, 2013, pp. 126–135.

[Che+13] Lingchen Chen et al. "A survey on hand gesture recognition". In: *Computer Sciences and Applications (CSA), 2013 International Conference on*. IEEE. 2013, pp. 313–316.

[Han+13] Jungong Han et al. "Enhanced computer vision with microsoft kinect sensor: A review". In: *IEEE transactions on cybernetics* 43.5 (2013), pp. 1318–1334.

[Pal+13] Jose Manuel Palacios et al. "Human-computer interaction based on hand gestures using RGB-D sensors." In: *Sensors (Basel, Switzerland)* 13.9 (Jan. 2013), pp. 11842–60.

[Sen13] Prime Sense. "NITE Algorithms". In: *PrimeSense NITE Algorithms 1.5*. Feb. 2013, pp. 1–3.

[Guy+14] Isabelle Guyon et al. "The ChaLearn gesture dataset (CGD 2011)". In: *Machine Vision and Applications* 25.8 (2014), pp. 1929–1951.

[Iba+14] Rodrigo Ibanez et al. "Easy gesture recognition for Kinect". In: *Advances in Engineering Software* 76 (2014), pp. 171–180.

[Wu+14] Xingyu Wu et al. "View-invariant gesture recognition using nonparametric shape descriptor". In: *Pattern Recognition (ICPR), 2014 22nd International Conference on*. IEEE. 2014, pp. 544–549.

[Bar+15] Lorenzo Baraldi et al. "Gesture Recognition using Wearable Vision Sensors to Enhance Visitor's Museum Experiences". In: *IEEE Sensors Journal* 15.5 (2015), pp. 2705–2714.

[RA15] Siddharth S. Rautaray and Anupam Agrawal. "Vision based hand gesture recognition for human computer interaction: a survey". In: *Artificial Intelligence Review* 43.1 (2015), pp. 1–54.

Flexible navigation through a multi-dimensional parameter space using Berkeley DB snapshots

Dimokritos Stamatakis
Brandeis University
415 South St
02453, Waltham, MA
United States
dimos@cs.brandeis.edu

Werner Benger
AHM Software GmbH
Technikerstrasse 21a
A-6020 Innsbruck,
Austria
w.benger@ahm.co.at

Liuba Shrira
Brandeis University
415 South St
02453, Waltham, MA
United States
liuba@cs.brandeis.edu

ABSTRACT

The concept of a visualization pipeline is central to many applications providing scientific visualization. In practical usage scenarios, when the pipelines fuse multiple datasets and combine various visualization methods they can easily evolve into complex visualization networks directing data flow. Creating and managing complex visualization networks, especially when data itself is time-dependent and requires time-dependent adjustment of multiple visualization parameters, is a tedious manual task with potential for improvement. Here we discuss the benefits of using Berkeley Database (BDB) snapshots to make it easier to create and manage visualization networks for time-dependent data. The idea is to represent visualization network states as BDB snapshots accessed via the widely used Hierarchical Data Format (HDF5), and exploit the snapshot indexing system to flexibly navigate through the high-dimensional space of visualization parameters. This enables us to support useful visualization system features, such as dynamic interpolation of visualization parameters between time points and flexible adjustments of camera parameters per time point. The former allows fast continuous navigation of the parameter space to increase animation frame rate and the latter supports multi-viewpoint renderings when generating Virtual Reality panorama movies. The paper describes how the snapshot approach and the new features can be conveniently integrated into modern visualization systems, such as the Visualization Shell (Vish), and presents an evaluation study indicating that the performance penalty of this convenience compared to maintaining visualization networks in HDF5 files is negligible.

Keywords

Scientific Visualization, Big Data, HDF5, Database Snapshots, Parameter space exploration

1 INTRODUCTION

Creating high-quality scientific visualizations of large time-dependent datasets can be time-consuming for the scientists. In data-flow based visualization systems this process can involve step by step creation of manually tuned instructions in the form of visualization pipelines (networks) that describe the flow of data from sources to sinks and are the basis elements of more complex visualization networks [1].

For large observational or computational data sets, common in todays systems, the list of visualization instructions can be long and different data points may need different structures, e.g. direct different number of cameras representing different validation perspectives, like in Virtual Reality environments.

Current visualization systems do not provide satisfactory support for managing long varying visualization instruction lists. For example, visualization systems store lists of instructions in spreadsheets, but the rigid structure makes it hard to accommodate instructions with varying number of parts [2]. Instruction lists are represented by specifying transformations that map each instruction into its successor instructions [3], which provides instruction derivation provenance but makes it hard to flexibly navigate through the visualization starting from arbitrary data point of interest.

To better support the large time-dependent datasets, visualization systems need to provide features that make instruction management more flexible and easier to manage. For example, it would be beneficial if the visualization systems could support visualization parameter interpolation between different time steps (similar to dead reckoning in video games or any non-linear video editing software such as Adobe

Premiere [4]) to reduce creation effort by avoiding storing redundant instructions. Furthermore, dynamic parameter interpolation would also allow to achieve higher animation rates or provide coarser views as needed during the visualization. Many current systems however do not support interpolation.

This paper discusses the benefits of using database snapshots to improve visualization system support for time-dependent data. Specifically, we describe how we extended the Visualization Shell (Vish) [5] using Retro [6], a snapshot system for the Berkeley Data Base (BDB) [7] to provide dynamic interpolation and flexible navigation for lists of visualization networks for time-dependent data in HDF5 format [8]. Our system represents lists of network states as a list of network versions stored as successive BDB snapshots. This representation can easily accommodate instructions with varying structure. We can take advantage of the snapshot indexing system to navigate the network versions and can support flexible traversals of the high-dimensional visualization parameter space stored in snapshots from arbitrary data points. The snapshot representation also makes it easy to support dynamic interpolation of parameters between the time points.

Our visualization system uses the portable HDF5 format to represent both the time-dependent data and the visualization networks. The time-dependent data is stored via HDF5 library in a file system well-suited for the large files. A natural question arises, why not store visualization networks in HDF5 files as well. To this effect, we have also experimented with a system that stores visualization networks directly in HDF5 files, instead of snapshots, implementing the flexible navigation manually, instead of relying on snapshot indexing. Using the snapshot system however was considerably simpler. Moreover, an important benefit of using database snapshot system is providing a reliable transactional storage for visualization networks that represent a substantial scientist time investment. By using the transactional snapshots we benefit from an automatic recovery of the visualization network structures after an early or involuntary program termination, something that would need to be achieved with complex ad-hoc recovery methods in a system that stores the networks in HDF5 files.

Our system therefore uses two different backend storage systems, HDF5 files for time-dependent data, and BDB snapshot system for the visualization instructions. Such approach, combining multiple storage systems each optimal for intended data use is becoming increasingly common in the new generation of big data systems due to the realization that no single storage system is optimal for the rich set of data comprising todays big data systems [9]. Our implementation takes advantage of the recently introduced HDF5 VOL software layer [8] that allows applications to access HDF5 data objects in different storage backends. For our system, we have implemented a new VOL that provides access to BDB snapshots, allowing the Vish system to seamlessly manipulate data and visualization metadata through the same HDF5 API.

A visualization system must perform well. We have conducted a study that evaluates the performance of our snapshot based Vish system and its new interpolation and navigation features, using micro benchmarks. We have also compared our system performance to the design that stores both data and visualization instructions in the HDF5 files. The measurement results indicate that our snapshot based system performs well, and the performance penalty we pay for the simplicity of implementation and reliability compared to storing instructions in HDF5 files is acceptable.

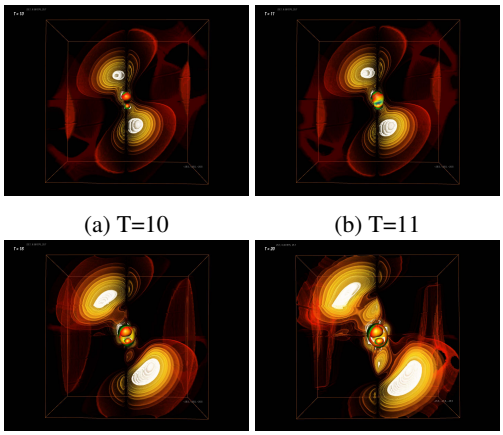
Our main contributions in this paper are the following:

- A simple database snapshot-based approach for managing visualization metadata for time-dependent HDF5 datasets
- Design and implementation of the BDB VOL plugin for HDF5
- Design and implementation of the Interpolation component in Vish visualization system
- Performance evaluation study of our snapshot-based approach and the interpolation feature

The rest of the paper is organized as follows. Section 2 describes a concrete scenario that motivated our work. Section 3 discusses related work, Section 4.1 explains the software structure of Vish system, the context of our work, Section 4.2 describes the interpolation feature, Section 4.3 highlights the salient points of our implementation, Section 5 presents the performance study and Section 6 our conclusions.

2 MOTIVATION

Solving Einstein's equations on supercomputers [10] is a grand challenge involving many institutions and generations of scientists. These efforts have culminated by the recently announced detection of gravitational waves. Numerical simulations of black hole collisions as the strongest sources are essential for the proper analysis of the detected signals. A milestone in numerical relativity was the first fully three-dimensional simulation of a grazing collision of two black holes [11]. Fig. 1 is showing four time steps from this simulation. Shown is the real part of the complex Newmann-Penrose pseudoscalar Ψ_4, an indicator of the outgoing gravitational radiation field, as elaborated in more detail in [12]. Visualizations like those are not only esthetically pleasing [13], but also important for scientific development to assess the quality and features of



(a) T=10 (b) T=11

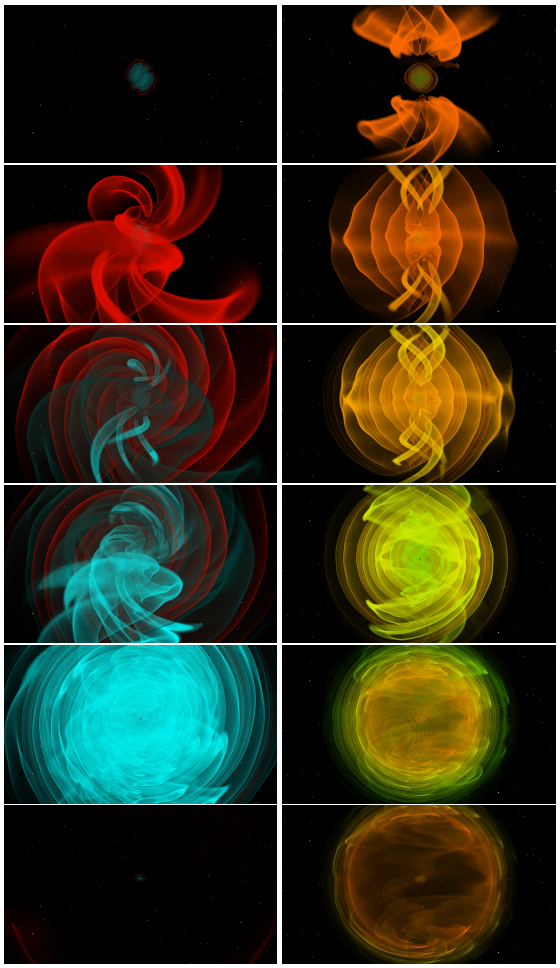
(c) T=16 (d) T=20

Figure 1: Burst of gravitational waves emerging from colliding black holes

big data. The first confirmed detection of gravitational waves [14], known as GW1509014, was simulated at the Max-Planck Institute for Gravitational Physics according to the observed collision parameters. This simulation produced a dataset of 400GB of binary data of much higher detail and precision than the 1999 dataset, particularly also covering a significantly longer time range lasting several orbits. Due to the nature of these astrophysically violent events the dynamic range of the data is huge, in space and time. While the 1999 dataset could be covered with parameters of constant range, trying the same approach on the 2016 dataset does not yield pleasing results, as shown in Fig. 2a: Initially the radiation is so weak that it is hardly visible at all, but its strength increases rapidly to emerge like a "flash" during a very small amount of time, leaving only residual radiation of a "wobbling" rotating black hole. This residual radiation fades away quickly at low intensity to ultimately form a non-radiating Kerr black hole.

The situation is similar to high dynamic range rendering: The output medium (images with 8 bits of intensity for each color) just cannot cover the entire range of the input data (orders of magnitude). In this case of evolving data a more suitable range can be found at each time step. Determining this range computationally is difficult though because of the wave-like nature of this astrophysical process leading to visually disturbing oscillations. This leaves manual adjustment of the data mapping range for color-coding as the only option, similar to controlling animations according to a movie director's intention via any non-linear video editing software. With the ability to fine-tune any parameter of a visualization network over time, we can thus extract the maximum structural information out of time-dependent data at the cost of less quantitative assessment abilities.

Another approach to cover high dynamic range is a global transformation such as computing the logarithm, as demonstrated in Fig. 2b. This approach worked in



(a) (b)

Figure 2: (a) Evolution of $\Re\Psi_4$, positive components in red, negative in cyan. (b) Evolution of $\log|\Re\Psi_4|$, covering a wider range at the cost of signature and detail.

this specific situation to provide an overview of the dataset without manual interaction at the loss of signature information – we can only see the absolute value of $\Re\Psi_4$, but not if its positive or negative, which roughly corresponds to stretching or compression of spacetime at the very location.

So while specific workarounds were able to yield visually pleasing results for the particular application scenario, a more systematic approach for arbitrary fine-tuning similar to professional video editing systems is desirable.

3 RELATED WORK

Visualization data management systems support versions of visualization graphs for different reasons and correspondingly use different approaches. The use of a visualization pipeline is pretty much a core standard among software frameworks for scientific visualization. Paraview[15] and Visit[16] are based on the Visualization Toolkit VTK[17]; OpenDX[18] and

Amira/Avizo[19] are other widely used frameworks. Vish system [5] is an independent visualization system with a much smaller user community, but comes with the most systematic approach to deal with visualization concepts while being academically open-source and a comparable small codebase; it is therefore the framework of our choice for our work, as described in 4.1.

VisTrails [20] is more of a management system to visualization systems than a visualizion framework itself. It uses executable XML specifications to generate sequences of visualization network graphs (Vistrail specifications) that can vary in structure and provides efficient runtime that allows efficient incremental visualizations along the time line. This method supports regular structure and parameter variations between networks at different time points but does not support flexible time and data dependent variations supported by our approach. VisTrails stores graphs using SpreadSheets so when the schema of the graph evolves, the history needs to be reformatted. Our snapshot approach supports arbitrary evolving network schemes without reformatting. The VisTrails data management system [3] represents specifications corresponding to successive data points by storing operations that transform one network into another to support derivation provenance. The representation requires to apply the full operation history to visualize a given data point making it inconvenient to traverse the version graph from arbitrary points. In contrast, our snapshot approach allows to navigate visualizations from arbitrary time points and can support provenance programmatically by storing transformations as additional attributes in snapshots.

ModelDB [21] manages a branching version history of machine learning models and visualizations aimed at exploration of alternative parameter configurations and like VisTrails specifies transitions using operations to support provenance. Our snapshot based approach, specialized for time-dependent data, currently only supports linear version histories.

Polystore [9] is a new generation scientific data management system that uses multiple storage backends to optimally accommodate data of different types, and implements an integrating layer to provide a unifying access to the different data parts. We adopt a similar approach by storing time-dependent data and visualization metadata in different storage backends, and take advantage of the HDF5 VOL infrastructure to add the snapshot system backend by implementing a new VOL.

4 SYSTEM DESIGN

This section describes our approach to managing visualization instructions for time-dependent data, and highlights the salient points of our implementation. We start by summarizing our requirements motivated by the use case of gravitational wave data visualization. To support visualization of time-dependent data a system needs to support:

- incremental creation of time-dependent visualization instructions, allowing scientists to conveniently adjust parameters for different time points, and exploit parameter interpolation to save effort when parameter changes can be computed.
- varying visualization instruction structure (e.g. extra cameras, or light sources) to offer customized views for different time points.
- adjustable frame rate (higher to lower) during visualizations using dynamic parameter interpolation.
- flexible navigation through the visualization parameter space from arbitrary time points.
- reliable persistent storage for instruction lists protecting scientist time investment in the presence of system crashes.

We describe below how our system design satisfies these requirements. We start by briefly explaining the features of Vish visualization environment, the context of our work. We then explain how we use a snapshot system to create and navigate instruction lists and present the design of the parameter interpolation feature. We then describe how we integrated the snapshot system backend into Vish.

A general view of the software layers in our system can be seen in Figure 3, where we show the Vish Visualization system (described in 4.1) and how its several components use the HDF5 library. First, we have the existing visualization module which is responsible for visualizing datasets aimed by networks. Then, we developed the parameter interpolation module, described in Sec. 4.2, and finally the dataset and network accessing modules allowing access to HDF5 data in native HDF5 format and BDB VOL format, respectively.

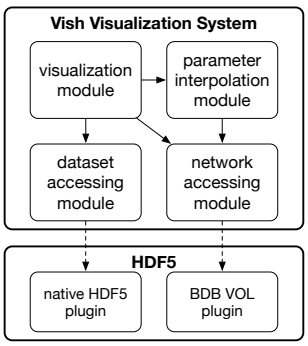
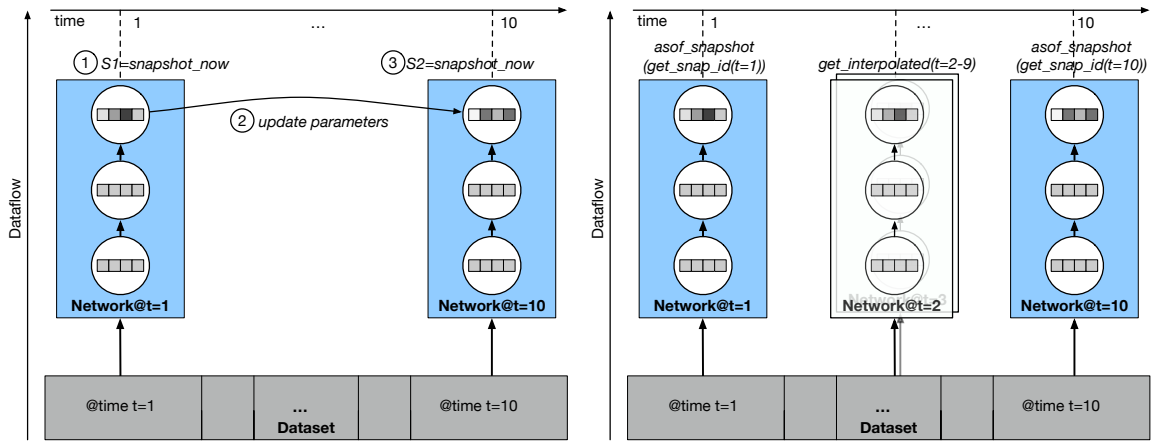


Figure 3: The software layering of our system

4.1 Vish

The Vish Visualization Shell [5] is a software environment specifically designed to “make everything a plugin”. It manages all its plugins functionally through



(a) Create a sequence of visualization pipelines for time-dependent data only for the desired time steps

(b) Display time-dependent data using the sequence

Figure 4: The process of dynamic parameter interpolation

a small kernel of abstract object interfaces which are used to implement graphics, user interface, I/O, or object relationships itself. This approach allows for modular well-encapsulated components that can be implemented independently of each other. The concept of a visualization pipeline and the resulting graphs for practical applications are built into this minimalistic object management framework. It can then be managed via a runtime-loaded graphical user interface, for instance via visual programming, or to interface any language that supports scripting.

As a reference implementation, Vish comes with its own, minimalistic scripting language that allows to store and load a visualization's network state, while remaining human-readable and human-editable. It is however not optimized for performance since parsing text inherently comes with a performance overhead. Using the Vish plugin architecture, it is straightforward to implement an alternative way of loading and storing a visualization network state using another format such as HDF5, which then avoids a parsing overhead due to its self-descriptive binary nature. This is the format we use in our work for storing visualization networks.

Even more, a visualization network's state can be managed while loaded by some Vish object itself. A graphical user interface is nothing else than a plugin providing a Vish object with such management functionality. Via the Vish kernel API, objects, their parameters and connections are exposed in an abstract way to allow generic interaction. We can use this very functionality to also produce new network states from existing ones, for instance via interpolation as explained below.

Time is supported in various Vish modules as a continuous floating-point quantity, leaving the notion of "time steps" to be implemented locally to each data set. This way also non-equidistant time steps, such as produced by the CFL condition in numerical simulations [22], are taken care of. Consequently, indexing data given at discrete time steps by a continuous time value requires data interpolation. When producing an animation with a given frame rate, the continuous time is subsampled by discrete steps again, however, independent of the time steps of the original data. We can hereby smoothly fuse data from different sources with different time discretizations. The Vish user interface contains plugins to sample the continuous time parameter space and produce discrete time steps for animation. We use this feature to navigate time points for network states and data.

4.2 Parameter Interpolation

Visualization of time-dependent datasets might require adjusting parameters other than time as visualization proceeds along successive time points in the data set. In this case, we want to keep track of different states of the network graph corresponding to different time points. In order to easily manage these states, we decided to use a Database snapshot system that provides a simple interface for creating and accessing data versions as database snapshots. We store the visualization networks in BDB [7], a key-value database, and manage network states using Retro [6], a snapshot system for BDB. Retro supports an easy implementation of the simple network state management workflow where a scientist visualizes the current data point using the current network state, adjusts the network state parameters as needed, stores the new network state by taking a snapshot of the adjusted state, and proceeds to the next time point. However, creating a separate network state manually for each key frame is time consuming and maybe unnecessary if the parameter changes can be interpolated automatically. Thus, we decided to add an option for dynamic parameter interpolation, which will calculate the missing intermediate parameter values and remove the burden from the scientists. This

can be easily implemented with our snapshot approach, since Retro allows programs to access snapshots and compute with data stored in snapshots the same way as with data stored in the database [6]. Thus our system can create only needed states and during visualization load the two corresponding consecutive snapshots and then calculate the desired network parameters dynamically for the individual intermediate time points.

Figure 4 illustrates the interpolation workflow. In Figure 4a we show the process of creating snapshots for a time-dependent dataset, representing an animation. We see the states of the visualization network graph at each time point, representing directions for how to visualize at that point. Each node in the graph represents a visualization object, which disseminates data from the dataset (source) to the sink. Within each object we see the object's parameters, with different color representing different parameter value. Arrows point to the direction of the data flow. Initially, we have the visualization network at time point $t = 1$ which points to the dataset at time point $t = 1$, and then we take a snapshot so that to store the network state at that point using the snapshot system API operation `snapshot_now`. This operation takes a snapshot of the current state and returns a snapshot identifier (`S1` in this case). We enhanced the implementation to store a persistent mapping from time point to snapshot identifier (`get_snap_id(t=x)`), so that to correlate each snapshot with a time step. Then, moving at time point $t = 10$, we update some parameters of the very first node in the graph (different color boxes) and when visualization looks good, we call the `snapshot_now` again for time $t = 10$, which returns snapshot identifier `S2`.

Simply displaying the network as of $t = 1$ for time points $t = 2$ to $t = 9$ might cause the visualization to change rapidly when reaching time point $t = 10$ which is undesirable for animations. Thus, we will use interpolation to smooth it. Figure 4b shows the process of loading the states of the visualization network graph from snapshots and displaying, resulting in playing the animation. For each time point, the visualization system has to check the time-snapshot mapping whether there is a corresponding snapshot and if not, interpolate parameters after loading two consecutive snapshots. At time $t = 1$, the system calls the Retro operation `asof_snapshot(get_snap_id(t=1))`, where `get_snap_id(t=1)` returns `S1` and `asof_snapshot(S1)` loads the snapshot with identifier `S1`. Thus, we load the snapshot corresponding to time point $t = 1$ providing the visualization network as it was at the time we took the snapshot.

Since time points $t = 2$ to $t = 9$ do not have corresponding snapshots, they are remapped to time $t = 10$ allowing us to load snapshot `S2`. Once we have parameter values for $t = 1$ and $t = 10$, we can calculate values for $t = 2-9$ using linear interpolation such that for each parameter A given at discrete time steps t_0 and t_1 we can compute its value as a smooth function

$$A(t) = A(t_0)(1-\tau) + A(t_1)\tau \quad , \tag{1}$$

where τ is the relative time, i.e. $\tau = (t-t_0)/(t_1-t_0)$ with $t_0 \leq t \leq t_1$. Higher order interpolations such as used on cubic splines [23] are possible as well, but require more snapshots to be accessed and evaluated, and a more complex time-to-snapshot mapping. We do the same for all intermediate time steps that we don't have a corresponding snapshot.

Of course, interpolation can also be used at creation time to create one snapshot per time point. This approach simplifies time to snapshot indexing but potentially stores a much larger number of snapshots and also does not allow flexible coarser or finer grained interpolations. The only case it could be beneficial is if the interpolation is effortsome, so storing snapshots for later playback would be faster. Thus, we believe the dynamic interpolation approach is preferable. Our performance evaluation considers both approaches in Section 5.

4.3 Berkeley DB VOL plugin

The current version of the Vish visualization system supports HDF5 format for both computational or observational data objects and visualization network objects, storing both data and networks in the HDF5 file system, and accessing them via the HDF5 library. This section describes how we extended the HDF5 library in the visualization system to support managing versioned visualization network objects in a snapshot system.

Since the native HDF5 file system does not support versioning, we had to provide versioning ourselves. We had two goals for our design. We wanted to provide versioning in a light way manner without modifications in the HDF5 codebase, and we wanted to allow the visualization system to continue use the HDF5 API for data and networks. Our design builds on a recent feature in HDF5 called Virtual Object Layer (VOL) plugins that allows the implementation of custom storage back-ends. A VOL plugin is a seamlessly connected component of the HDF5 library that is responsible for storing and accessing HDF5 data containers in a particular storage back-end.

We support versioning of visualization networks using a VOL plugin that stores network states represented as HDF5 objects in a backend that supports versioning. Our versioning back-end storage is Retro system that provides snapshots for Berkeley DB [6]. Retro allows easy versioned network state creation, simple version indexing and supports easy computation with versioned states e.g. to support dynamic interpolation feature explained earlier. Importantly, Retro provides reliable crash consistent storage for versions thus satisfying

```
Current-state queries are unchanged by Retro
    results ← { gid = H5Gopen2(fid, node_name, ...);
                attr = H5Aopen(gid, attr_name, ...);
                H5Aread(attr, mid, buf);
                ... }

Applications may declare snapshots at any time and
get back a snapshot identifier
    S ← snapshot_now

As of queries are delimited with the snapshot identifier
    results ← asof_snapshot S { gid = H5Gopen2(fid, node_name, ...);
                                attr = H5Aopen(gid, attr_name, ...);
                                H5Aread(attr, mid, buf);
                                ... }
```

Figure 5: Example of using the Retro API to retrieve attributes from a visualization network

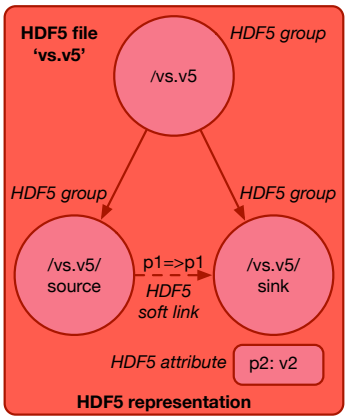
our requirement for consistent recovery of complex network graphs stored in memory during a crash. We can obtain versioning by simply implementing a new VOL plugin. Moreover, since networks are stored in Retro in HDF5 format, the visualization system can continue to manipulate them using the HDF5 API. This approach achieves both our design goals.

Figure 5 shows an example of using the Retro API to retrieve an attribute of a node from a visualization network stored as HDF5 objects. The query opens a group with name `node_name` from the HDF5 file with identifier `fid`. Then, it opens an attribute with name `attr_name` and reads the attribute value in the specified buffer `buf`. We create snapshots by using the `snapshot_now` operation which returns a snapshots identifier. In order to perform a past state query in state S, we simply wrap the same query with `asof_snapshot S`.

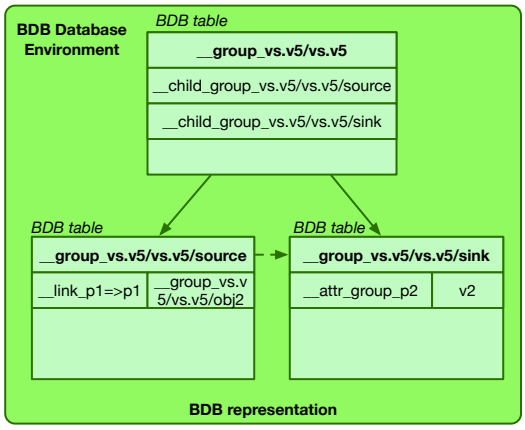
We developed a new HDF5 BDB VOL plugin to intervene between regular HDF5 API calls and Retro. The VOL plugin developer is responsible for mapping the HDF5 data model to the schema of the backing store. In our case, BDB schema is simple and consists of tables and key/value pairs within the tables. Thus, we can organize HDF5 objects as separate BDB tables and store their attributes as key/value pairs. Connections between HDF5 objects are represented as pointers between BDB tables.

We represent visualization networks as HDF5 objects, as shown in Figure 6a. In this example network, nodes *source* and *sink* are stored as subgroups under */vs.v5* group. HDF5 soft links are used for node connections and HDF5 attributes to store node parameters. The mapping from HDF5 objects to BDB tables in this example is illustrated in Figure 6b.

We are using Berkeley DB transactions for all accesses to networks, so that to support recoverability and crash consistency. We extended therefore HDF5 user API with operations to begin and end transactions, and with Retro API operations `snapshot_now` and `asof_snapshot`. Since Retro assigns a number as



(a) Network representation in HDF5



(b) Network representation in BDB

Figure 6: Visualization network representations

an identifier for snapshots, we preserve the mapping from time steps to snapshot identifiers using a separate mapping table stored in BDB. BDB and by extension Retro is optimized for relatively small data accesses and would not be efficient backend for large scientific time-dependent data. Our design therefore exploits the flexibility of VOL and continues to store data objects in the native HDF5 file system.

5 PERFORMANCE EVALUATION

We have implemented the versioned metadata module in Vish and conducted an evaluation study. The goal of the study is to evaluate the cost of managing versioned visualization networks using BDB snapshots, and compare it to a simple alternative design where network versions are stored in HDF5 files. In all cases the time-dependent datasets are stored in HDF5 files. Our experiments use Vish to visualize the time-dependent data set, measuring the performance of creating and loading network versions to visualize successive data points.

Our experimental setup consists of a machine with a 6-core i7 at 4.6GHz, equipped with 32GB memory, 6GB

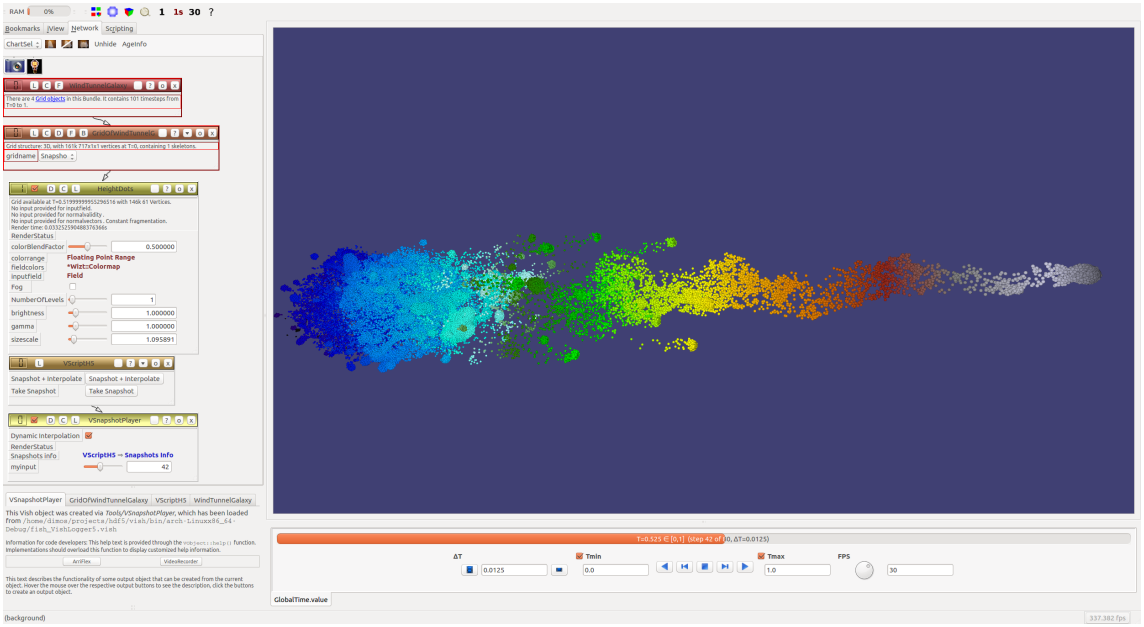


Figure 7: Displaying a time-dependent dataset on Vish, by loading snapshots and performing dynamic interpolation

GPU Nvidia GTX 1060 and an SSD drive. It is running Ubuntu Linux 16.10 with the Retro Berkeley DB C API 5.3, HDF5 1.9.3 with the VOL support, and the developer's edition of the Vish Visualization Shell. We used several datasets for our experimental setup ranging to several GB in size and used up to 81 time steps.

Our versioned metadata module using Retro/BDB implements the design we described in earlier sections. When the user performs modifications to the network through the GUI, the modifications are forwarded to the Retro/BDB VOL that performs them with a transaction that writes them into the BDB log. Later, when user creates a version (a snapshot) or terminates visualization, the transaction commits, flushing the log containing these modifications at commit time, and eventually writing the modifications lazily at low cost to the Retro store. Thus, all committed modifications are preserved after a system crash.

For the versioned metadata module using native HDF5 files we used a simple-minded design (called Native HDF5) that stores consecutive versions of the network as different HDF5 groups. HDF5 groups are analogous to directories in a typical file system. They may contain other groups (subgroups), attributes, datasets, etc. When creating a version, the Native system creates a new HDF5 group, writes it in its entirety to the new group, and associates that group with a unique identifier that allows to access the version later using a simple index to support interpolation. Since both the new group and the index are written to the file system at version creation time this approach does not provide crash consistency. Obviously, the simple-minded approach stores redundant data if only few parameters change between versions. A diff-based encoding and additional indexing would provide a more compact solution but would require more substantial development effort, including a more complex recovery procedure after crash to avoid version and index corruption.

Our experiment emulates a user workflow similar to Figure 4, where a user creates a versioned visualization for a time-dependent data set, starting with an initial network specification for the first data point. When visualizing at a certain time step the user adjusts the visualization parameters, stores them by creating a version (a snapshot, or a new group) and repeats these steps for all data points. At any point, the user can visualize the data for any time step, or play all time steps as an animation. We used a 2.8GB time-dependent data set from an astrophysical simulation [24] using 81 time steps. Figure 7 shows a single visualization time step of that simulation obtained in our experiment by loading snapshots and performing dynamic interpolation.

Our experiment measures the basic cost of writing the consecutively created versions (not including think time) and of subsequently loading the versions and visualizing consecutive data points, using either the simple versioning module with HDF5 (native HDF5), or the Retro/BDB VOL snapshot system. Our experiments also evaluate the dynamic interpolation feature that allows to only create network versions for selected data points omitting intermittent points and interpolating them dynamically at display time as seen in Section 4.2. Additionally, we also evaluate an alternative approach with one-to-one snapshot to time step mapping, which interpolates missing values between manually adjusted versions at creation time, rather than at display. This creates a network version for each data time point avoiding the need to interpolate

at run time. As we explained, the dynamic interpolation is preferred, but the one-to-one mapping approach can use simpler indexing.

In Figure 8 we see the cost of creating a snapshot by using the native HDF5 approach and the Retro/BDB VOL, when doing one-to-one mapping and dynamic. Initially we observe that the native HDF5 approach is costlier in snapshot creation, as expected. This is because it writes the entire network in HDF5, regardless of which parameters are updated. It takes 35 ms to create a snapshot when doing one-to-one mapping, and 44 ms when doing dynamic interpolation. The one-to-one mapping is slightly faster than the dynamic, since the creations are performed in a loop, without involving a user interface. We see better performance for Retro because this design only stores the updated parameters, rather than the entire network and keeps track of different versions natively. Note Retro costs reflect the writing of the transactional log, which includes writing to disk all updates to the committed version at commit. As we can see from Figure 8, Retro only takes about 0.5ms to create a snapshot in one-to-one mapping and less than 4 ms in dynamic, including both the creation of a new snapshot and the transaction commit. It is slower in dynamic, per version, since we perform more modifications between two snapshots. In general, even the 44 ms to create a version with native HDF5 are not perceptible for a user in an interactive setting. However, the one-to-one mapping that creates many versions, can cause noticeable latencies of up to a second with our data set.

Figure 9 shows the overhead of loading from a version when using the native HDF5 or Retro with one-to-one mapping and dynamic. Loading from a snapshot involves opening the snapshot for a specified time step and reading only the interpolated parameters. This avoids accessing the entire network but also incurs the overhead of snapshot indexing in order to find the specified snapshot. In this case, the native HDF5 is faster because while it writes the entire version it only has to open the specified version (from the corresponding HDF5 group within the same HDF5 file) and get the required parameters.

When loading a snapshot created with one-to-one mapping, we saw a response time of 565 µs on average, compared to Retro which took 663 µs. Next, when loading snapshots in dynamic, both methods are faster because there are less snapshots to manage. Native HDF5 takes 271 µs and Retro 364 µs. Obviously, version loading time is typically more important than creation since in the common case we expect versions to be created manually but displayed automatically. Nevertheless, we do not expect automatic visualization to be limited by the metadata accessing time, since the typical expected bottleneck is reading the large time-dependent

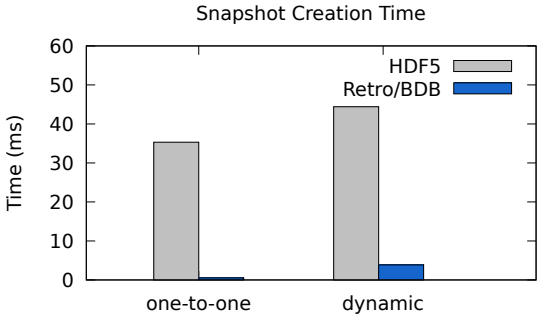


Figure 8: Time to create a snapshot of the current visualization network under different configurations.

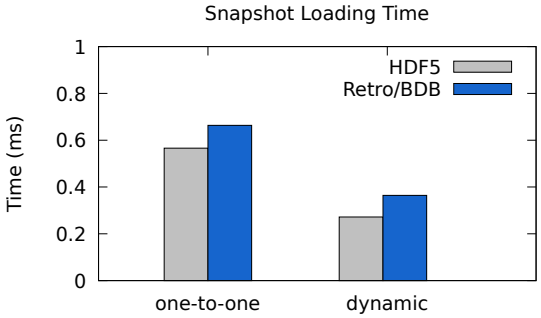


Figure 9: Time to load from snapshot and update the visualization network under different configurations.

datasets. In both cases of native HDF5 and Retro, we have around 0.5 ms response time to bring metadata for displaying a frame. This is negligible compared to the time to load the big data, and does not limit the frame rate.

6 CONCLUSION

We have presented a new, database snapshot system based method for managing visualization instructions for large time-dependent scientific datasets that are not well supported in current visualization systems. We explained how a simple programming model provided by the snapshot system makes it easy to manage versioned visualization metadata and to provide new labor-saving visualization system features such as version interpolation that reduce the manual effort needed to develop visualizations. We have described how we implemented our approach in the Vish visualization system and presented experimental results using a small data set from an astrophysical simulation indicating satisfactory performance while providing better reliability guarantees. Future work is using our approach for developing visualization for the large gravitational waves dataset.

Acknowledgment

This work is partially supported by National Science Foundation awards CNS-1318798, IIS-1251037, and IIS-1251095. We thankfully acknowledge Nikos Tsikoudis for his support with the Retro BDB.

7 REFERENCES

[1] R. B. Haber et al. Visualization Idioms: A Conceptual Model for Scientific Visualization Systems. In *Visualization in Scientific Computing*. 1990.

[2] Jankun-Kelly et al. A Spreadsheet Interface for Visualization Exploration. In *Proceedings of the Conference on Visualization '00*, VIS '00, pages 69–76, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[3] Steven P. Callahan et al. VisTrails: Visualization Meets Data Management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 745–747, New York, NY, USA, 2006. ACM. doi:10.1145/1142473.1142574.

[4] Adobe Premiere. URL: http://www.adobe.com/products/premiere.html.

[5] Werner Benger et al. The Concepts of VISH. In 4^{th} *High-End Visualization Workshop, Obergurgl, Tyrol, Austria, June 18-21, 2007*, pages 26–39. Berlin, Lehmanns Media-LOB.de, 2007.

[6] Ross Shaull et al. A Modular and Efficient Past State System for Berkeley DB. In Garth Gibson and Nickolai Zeldovich, editors, *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014.*, pages 157–168. USENIX Association, 2014.

[7] Michael A. Olson et al. Berkeley DB. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '99, pages 43–43, Berkeley, CA, USA, 1999. USENIX Association.

[8] HDF5. Hierarchical data format version 5. http://www.hdfgroup.org/, 2009. The HDF Group.

[9] Jennie Duggan et al. The BigDAWG Polystore System. *SIGMOD Rec.*, 44(2):11–16, August 2015. doi:10.1145/2814710.2814713.

[10] Gabrielle Allen et al. Solving Einstein's Equation on Supercomputers. *IEEE Computer*, 32(12):52–59, December 1999. doi:10.1109/2.809251.

[11] Miguel Alcubierre et al. The 3D Grazing Collision of Two Black Holes. *Phys.Rev.Lett.*, 87, 2001.

[12] Werner Benger et al. Using Geometric Algebra for Navigation in Riemannian and Hard Disc Space. In Vaclav Skala and Dietmar Hildebrand, editors, *GraVisMa 2009 - Computer Graphics, Vision and Mathematics for Scientific Computing*. UNION Agency, Na Mazinach 9, CZ 322 00 Plzen, Czech Republic, 2010.

[13] Werner Benger et al. Visions of numerical relativity. In A.Gyr et. al., editor, *Proceedings of the 3d International Conference on the Interaction of Art and Fluid Mechanics (SCART2000)*, pages 239–246, ETH Zürich Switzerland, Feb 28 – Mar 3 2000. Kluwer Academic Publishers.

[14] Abbott B. P. et al. Observation of gravitational waves from a binary black hole merger. *Phys. Rev. Lett.*, 116:061102, Feb 2016. doi:10.1103/PhysRevLett.116.061102.

[15] Kitware, Inc. Paraview - open-source scientific visualization, 2010. URL: http://www.paraview.org/.

[16] DOE/ASCI. Visit, 2002-2010. URL: https://wci.llnl.gov/codes/visit/.

[17] Kitware. Visualization toolkit, 2005. URL: http://www.kitware.org/.

[18] Visualization and Inc. Imagery Solutions. Ibm open visualization data explorer, 2000-2006. URL: http://www.opendx.org/.

[19] D. Stalling, M. Westerhoff, and H.-C. Hege. Amira - an object oriented system for visual data analysis. In Christopher R. Johnson and Charles D. Hansen, editors, *Visualization Handbook*. Academic Press, 2005. URL: http://www.amiravis.com/.

[20] L. Bavoil et al. VisTrails: enabling interactive multiple-view visualizations. In *VIS 05. IEEE Visualization, 2005.*, pages 135–142, Oct 2005. doi:10.1109/VISUAL.2005.1532788.

[21] Manasi Vartak et al. ModelDB: A System for Machine Learning Model Management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '16, pages 14:1–14:3, New York, NY, USA, 2016. ACM. doi:10.1145/2939502.2939516.

[22] Stefano Nativi et al. Differences among the data models used by the geographic information systems and atmospheric science communities. In *In: Proceedings American Meteorological Society - 20th Interactive Image Processing Systems Conference. (2004*.

[23] Edwin Catmull et al. A Class of Local Interpolating Splines. In ROBERT E. BARNHILL and RICHARD F. RIESENFELD, editors, *Computer Aided Geometric Design*, pages 317 – 326. Academic Press, 1974. doi:http://dx.doi.org/10.1016/B978-0-12-079050-0.50020-5.

[24] Dominik Steinhauser et al. Simulations of ram-pressure stripping in galaxy-cluster interactions. 591:A51, 2016. doi:10.1051/0004-6361/201527705.