Self-Organizing Maps for Intuitive Gesture-Based Geometric Modelling in Augmented Reality

Benjamin Felbrich, Achim Menges
Institute for Computational Design and Construction
University of Stuttgart
Stuttgart, Germany
benjamin.felbrich@icd.uni-stuttgart.de

Gwyllim Jahn, Cameron Newnham
Fologram
Melbourne, Australia
gwyll@fologram.com

***Abstract*—Modelling three-dimensional virtual objects in the context of architectural, product and game design requires elaborate skill in handling the respective CAD software and is often tedious. We explore the potentials of Kohonen networks, also called self-organizing maps (SOM) as a concept for intuitive 3D modelling aided through mixed reality. We effectively provide a computational "clay" that can be pulled, pushed and shaped by picking and placing control objects with an augmented reality headset. Our approach benefits from combining state of the art CAD software with GPU computation and mixed reality hardware as well as the introduction of custom SOM network topologies and arbitrary data dimensionality. The approach is demonstrated in three case studies.**

Keywords - augmented reality; computer-aided design; parametric design; self-organizing maps

I. Introduction

The notion of utilizing a heads-up, see-through, head-mounted display to aid manual manufacturing processes dates back to the early 1990s [1]. Here the idea of overlaying the physical act of assembly with helpful data to guide the operator bears tremendous potential in much faster and less error-prone manufacturing processes. As hardware in the form of commercial headsets becomes more accessible to manufacturers, owing to – among others – the introduction of Microsoft's Hololens, this idea is making its way from fundamental research into actual applications and commercialized workflows. It is based on the utilization of predetermined digital information at hand, often provided in the form of a 3D model of the workpiece with the relevant information.

The creation of this digital information through complex geometric modeling, however, still largely remains within the realm of conventional desktop PCs and the modulation of geometry through keyboard, mouse, trackpad or more advanced 3d navigation devices like a "SpaceMouse" at best. Especially in creative fields like architecture, game or automotive design where demands in complex shape control span beyond accumulating and intersecting simple geometric bodies, but deal with complex doubly-curved surfaces, the creative process still largely depends on the classic workflow: coarse prototypes are being drafted through hand drawn sketches, then formalized in rough digital 3D models, and instantiated in the real world through models made from clay, foam or cardboard to give a realistic impression of the object. This process is repeated and refined, until the design task is considered complete and further production planning can take place. In the reverse manner, an object modelled in clay can be digitized through 3D scanning.

The work at hand aims to provide an alternative to this time and material-consuming approach by transferring the entire workflow of prototypic 3D modelling into the realm of augmented reality. Although the notion of using AR for 3D modeling is not new by any means, current technology often only allows for rather simple procedures such as moving, rotating, scaling simple objects or extruding planar surfaces. But more sophisticated means of geometry manipulation bear tremendous potentials for the creative community.

The Self-Organizing Map (SOM) is a neural network algorithm, which uses a competitive learning technique to train itself in an unsupervised manner. As opposed to other artificial neural networks SOMs use a neighborhood function to preserve topological properties of the input space - typically a two-dimensional grid in which the neurons are organized. They are capable of creating a simplified, ordered representation of multi-dimensional data and are often used for clustering, prediction, data representation, classification and visualization. Due to this topological nature, an adaption in the form of polygonal meshes stands to reason. Although other powerful surface modeling paradigms like NURBS modeling exist, SOM-based geometric modeling inherently offers the use of higher dimensional data to incorporate properties like color and circumvents further processing steps for triangulation in the display pipeline. Furthermore, due to the highly incremental nature in its convergence, additional geometric constraints such as mesh smoothing can be applied during runtime. The SOM's main drawback, the high computational cost in the evaluation phase, can be overcome by the use of highly parallelized state-of-the-art GPU computation as this article aims to show.

978-1-5386-9269-1/18/$31.00 ©2018 IEEE
DOI 10.1109/AIVR.2018.00016

II. Related Research

AR applications have recently become ubiquitous with software development kits for consumer mobile platforms such as ARCore and ARKit. Extensive research has been conducted to identify mixed reality applications within the architecture, engineering and construction industry [2] and a proliferation of recent literature demonstrates the use of mobile augmented reality for design visualization and construction review tasks [3] and for prototypic haptic design tasks [4]. The use of augmented reality interfaces to assist with design modelling and simulation has also been shown. These include the use of motion controllers to create tactile and gestural interfaces for parametric models [5] and projected AR to calibrate digital and physical simulations of material behavior [6].

Since the SOM algorithm was introduced by Teuvo Kohonen in 1982 [7] it found applications in various research fields wherever dimensionality reduction or mapping of large amounts of multidimensional data is of relevance. It ranges from structuring and visualizing vast amounts of weather data [8] to project prioritization and selection [9] or associating word meanings in language processing [10]. The benefits of an SIMD implementation of the SOM algorithm using CUDA was shown in [11].

Studies related to SOM in 3D modeling and adaptive meshing aimed at reconstructing a given geometry with a topologically ordered mesh [12]. Other CAD-oriented studies employ the SOMs clustering capability to treat design relevant information and / or produce an ordered range of related design objects such as building blocks [13]. Parametric design, the rule based, algorithmic branch of computer-aided design is especially eligible to benefit from the SOM paradigm as [14] alludes to. Here the interpretation of self-organizing data in a 3D modeling context exceeds the mere positioning of 3-dimensional vertices but serves as input for a more complex parametric procedure of geometry generation.

Other related methods for mesh-based 3D modelling such as free-form-deformation [15], mean value coordinates [16] or bounded biharmonic weights [17] are usually based on bending or deforming an already existing geometry. However, the SOM approach was chosen for its potential to generate a meaningful geometric solution, even when vertices are initialized randomly. This allows for modeling objects even with rather limited modes of interaction, as no complex objects have to be sketched manually up front.

III. SOM-Based Geometric Modelling

A. Adaptation of SOM Algorithm

In order to fully utilize the potentials of SOMs for 3D modeling, a number of restrictions imposed by classic SOM model had to be reconsidered: 1) the conventional topology of 2-dimensional square or triangle grids was replaced by an overall support for user-defined topologies. 2) the conventional use of 2d or 3d feature vectors, as they are suggested by 3D mesh modelling was generalized to use n-dimensional feature vectors. 3) Conventional SOMs serve to cluster large numbers of training vectors; on the contrary we propose the use of a small number of training vectors over very large number of neurons. Thus we're able to control the network's shape with relatively few training vectors, in the following referred to as "control objects". 4) Due to increased computational cost caused by serial network evaluation and weight processing, a CPU-based SOM implementation is practically inapplicable for intuitive real-time interaction with complex SOMs. Thus a highly parallelized implementation in CUDA was favored. These aspects are explained in more detail hereafter.

B. Graph Topology and Learning Function

In our starting setup SOM neurons are assumed to be vertices of a polygon mesh, their weights - respectively mesh vertex positions - are denoted $W \in \mathbb{R}^n$; further the mesh edges are interpreted as the synaptic connections E between neurons. The classic SOM learning algorithm is explained in [7]. Our adapted workflow is summarized as follows:

The neuron weights W are initialized according to a predefined starting configuration - e.g. a sphere shaped polygon mesh - along with a set of training vectors $C \in \mathbb{R}^n$, called control objects.

C. General SOM Learning Rule

A control object $C(t)$ is fed to the network by computing the Euclidean distance between $C(t)$ and all neuron weights W. The neuron closest to the control point is called the winning neuron; its index is u. Its weight W_u will be adjusted toward $C(t)$. All other neuron weights W_v in the network are also adjusted toward $C(t)$. The magnitude of change in W_v decreases with time and the neuron's grid-distance to the winning neuron with the following update rule:

$$W_v(s+1) = W_v(s) + \theta(u,v,s) * \alpha(s) * (C(t) - W_v(s))$$

With v being the index of the neuron to update, s being the index of the iteration and $\alpha(s)$ being a monotonically decreasing learning coefficient. $\theta(u,v,s)$ is the neighborhood function depending on s and the graph distance between neuron v and u.

D. Neighborhood Initialization

In the initialization phase we utilize the information about the predefined mesh topology to build and store a neighborhood tree. When defining the topology in the form of mesh faces, the scripting framework of our chosen 3D modeling environment Rhino3D allows direct access to vertex indices that are conjunct to a chosen vertex through a single mesh edge $e \in E$. These direct neighbor indices to vertex v are Ω_v. The grid-distances D_v for every vertex v to all other neurons is computed from Ω using following procedure:

1) *Store $v + \Omega_v$ as already mapped indices*
2) *Store Ω_v as the current neighborhood ring*
3) *Set current neighborhood level to 2*
4) *While the number of already mapped indices is smaller than the total number of neurons do:*

a) Get the directly connected neighbors Ω_i for each index i in the current neighborhood ring
  - For every direct neighbor index j in Ω_i check if it is already included in mapped indices or in the current neighborhood;
  - if it is not mapped yet, store current neighborhood level as the grid-distance d_{vj}, add j to mapped indices and to 'next neighborhood ring'

b) increase current neighborhood level by 1

c) reassign 'next neighborhood ring' to be 'current neighborhood ring' in the next iteration

The procedure terminates when all neurons in the SOM are mapped and works for arbitrary open and periodic (circular) topologies alike. Neuron v itself is stored as neuron neighborhood of level 0, its direct neighbors as level 1 and all other neurons in their respective neighborhood levels (fig. 1). The procedure is executed for every neuron concurrently in an individual GPU thread. The collection of mapped grid-distances D is of size $|W|^2$ and is stored in GPU memory for quick access during runtime. As the number of neurons is deliberately limited to 2^{16} and indices are stored as two-byte integers, the memory required to store D never exceeds 8GB of GPU RAM. However, this theoretical upper limit was never reached in our workflow. Furthermore the largest grid-distance between two neurons that was found in the entire network is stored as d_{max}. It is then used to scale the neighborhood function.

E. Learning Value Function

The commonly used Gaussian function was chosen to be the neighborhood function as it performed better than the Mexican Hat function in several tests. The general form of the Gaussian is:

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$$

Where a is the height of the bell curve's peak, b is the position of the center and c modulates the curve's width. To adhere to our learning goal, it is adapted as:

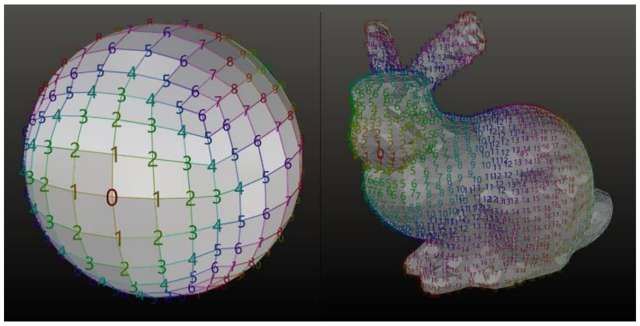

Figure 1. Levels of neighborhood for one mesh vertex (0): quad mesh sphere (left), mixed topology (right)

$$\theta(u,v,s) = e^{-\frac{d_{u,v}{}^2}{2c(s)^2}}$$

with $d_{u,v}$ being the grid distance between the winning neuron u and the neuron in question v. Here c represents the radius of neighbor neurons that are considered, also called the learning radius. It modulates the function's width according to the largest grid-distance d_{max} and the current iteration. The full width at half maximum (FWHM) for the Gaussian function [18] is:

$$FWHM = 2 * \sqrt{2 * \ln(2)} * \sigma$$

As we're only interested in the neighborhood radius, the half width at half maximum (HWHM) is more relevant to us. It is by our definition the upper limit of neighborhood propagation, the farthest grid-distance between two neurons d_{max}.

$$d_{max} = HWHM = \sqrt{2 * \ln(2)} * \sigma$$

Hence,

$$\sigma = \frac{d_{max}}{\sqrt{2 * \ln(2)}}$$

We finally scale the bell curve to get narrower with increasing iterations, to damp the neighborhood effect over time and allows convergence

$$c(s) = \sigma - \frac{\sigma * s}{S}$$

Where s is the current iteration and S the total number of iterations.

The learning rate $\alpha(s)$ serves to further scale down the movement with increased iterations to encourage convergence. Through comparative testing, we chose a hyperbolically decreasing learning rate relying on the initial learning rate $l_{start} = 0.25$ and a final learning rate $l_{end} = 0.03$. Thus

$$\alpha(s) = \frac{l_{start} * l_{end}}{l_{end} + \frac{(l_{start} - l_{end}) * s}{S}}$$

With $\theta(u,v,s)$ and $\alpha(s)$ defined we can define the product of these two modules as the learning value $L(u,v,s) = \theta(u,v,s) * \alpha(s)$ (fig. 2) and slightly change the update rule.

$$W_v(s+1) = W_v(s) + L(u,v,s) * (C(t) - W_v(s))$$

Similarly to the neighborhood topology, the results of $L(u,v,s)$ are computed during initialization and stored in GPU memory. Thus computation time during runtime is

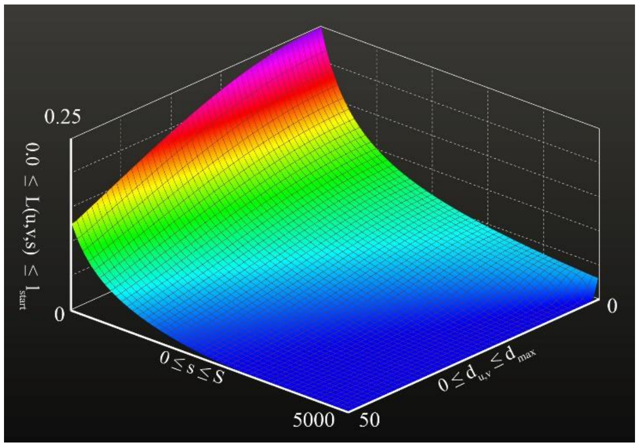

Figure 2. Learning value function $L(u, v, s)$ for maximum iterations $S = 5000$ and largest grid distance $d_{max} = 50$; the function produces learning values between 0.0 and $l_{start} = 0.25$; $l_{end} = 0.03$

minimized to simple memory access. With a size of a single precision floating-point number multiplied by $S * d_{max}$, the memory requirement is relatively small.

F. Implementation

As the scope of the proposed algorithm is narrowly defined, third-party libraries optimized for generality could be avoided in favor of a bare CUDA C/C++ implementation with little computational overhead. In addition to the SOM computation core, a thin layer in C++/CLI had to be established. It forms the connection between unmanaged code of the SOM core and the .Net CAD environment Rhino3d and Grasshopper3d in which the geometry display, interaction and connection to the Hololens is handled. It is integrated into the workflow in the form of a Grasshopper plugin. The SOM core's main components are: 1) an evaluation kernel to compute the Euclidean distances between training vectors and neurons. Here each training vector is assigned a single thread block and - along with all neuron weights - loaded into very fast shared memory. Each thread computes the distance between one training vector and one neuron weight vector and stores it in shared memory. A secondary reduction kernel finds the shortest distance for each training vector. 2) A movement kernel applies changes to the neuron weights based on the learning procedure described above. This step heavily benefits from the results of $L(u, v, s)$ being pre-computed and stored in GPU memory. Each neuron is handled by an individual thread.

Case study 1 (fig. 3) shows the characteristics of this approach: A three-dimensional space grid of 16 x 16 x 16 neurons is trained using eight three-dimensional training vectors. The precomputed grid distances D consist of the distance between each neuron pair, thus $16^{3^2} = 16{,}777{,}216$ multiplied by the size of a two-byte integer. This results in 32MB of storage required with d_{max} being 3 * (16-1). As we intend to interact with the model, S is deliberately kept small at 1000 to avoid long convergence times. The cycle time per weight update of the entire network including rendering time is 0.4ms without further processing and 0.5ms if an additional one-step Laplacian smoothing is applied [19]. As soon as a control object is modified, the training phase is restarted from the neurons latest positions. Thus the grid follows the moving control objects instead of retracting to its initial position.

IV. Computer Aided design in augmented reality

McNeel and Associates' Rhinoceros 3D is a computer aided design package designed to facilitate broad exploration of conceptual ideas through 3D modelling. Grasshopper 3D, a module-based visual programming extension for parametric modelling in Rhinoceros 3D, extends this capability by allowing users to dynamically generate geometry by "expressing a geometric model as explicit functions of a number of parameters" [20]. Fologram, a third-party extension for Rhinoceros 3D and Grasshopper 3D, was developed by the authors to enable designers to develop mixed reality applications using existing third-party Grasshopper 3D extensions and parametric models. Providing designers with a means to leverage their expertise with CAD modelling software improves the accessibility of mixed reality development to those within the architecture, engineering and design industries and eliminates the overheads associated with building interactive CAD models within game engines.

Fologram creates a bi-directional data stream that renders geometry in the CAD environment on the HoloLens, implements Vuforia's image target SDK to precisely align digital and physical spaces and provides access to HoloLens sensor data and gesture detection events as a parametric input source. Natural and intuitive interaction with parametric models can be facilitated by associating input parameters with the physical motions of a user within a mixed reality environment. For instance, the proximity of a user's hand to control vertices in the SOM model can be used to isolate and translate these vertices based on relative hand motion. This translation can occur at one-to-one scale in x, y and z axes simultaneously and enables the user to perform and evaluate design operations directly within the context of their physical workspace.

The concept of this software follows the visual programming principle. Relevant functions like gesture

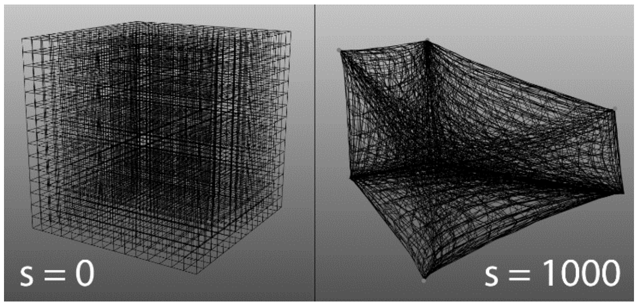

Figure 3. Case study 1: 16x16x16 nodes, S = 1000, $l_{start} = 0.3$, $l_{end} = 0.03$; additional 1-step Laplacian smoothing; convergence time: 500ms on an Nvidia GTX1080 Ti

recognition, spatially tracking the Hololens or sending geometric information to the augmented display are accessible through modules which can be connected at will. Along with already existing functions offered by the CAD environment, this open-ended design allows users to intuitively establish their own workflow.

V. RESULTS

The following case studies were processed using a desktop host PC with an Nvidia GTX1080 Ti and an Intel Core i7-8700K @ 4.7GHz. The visualization of and interaction with the model through the Hololens was enabled through a 2.4 GHz Wifi connection. As mesh information is sent to the Hololens asynchronously, the SOM computation on the host PC runs uninterrupted. This results in fast convergence but entails the risk of lag between the two model representations. Therefor the sizes of the following two case studies were chosen to be sufficiently complex for modeling, but small enough to be transmitted in real-time. Case study 2 (fig. 4, fig. 5) shows the modeling of a car geometry of 600 mesh faces, both quads and triangular. Each of the model's 602 nodes consists of their x, y, z position and a value for hue and saturation, thus forming a five-dimensional data set and allowing for node colors to be adjusted to the color of the nearest control object. While the control objects' colors were pre-defined, their x, y, z positions are changed through picking and dragging them around through a 'tap and hold' gesture. To identify the correct control object to drag, their x, y, z position along with the hand position are projected onto the plane of view. The control object that is closest to the hand in this projection gets selected; its display radius is enlarged to indicate to the user that it is active

This way the user can manipulate nine control objects, which are automatically mirrored against the xz -plane, ultimately resulting in 18 control object. This allows for an easy modeling of symmetric objects. Changing one control object's position triggers the model to be retrained, starting from its latest configuration. Thus the geometry barely changes in areas where near control objects haven't been altered, but deforms heavily when nearby control objects were displaced. Through an 'air tap' gesture, the retraining from the network's initial ellipsoid configuration can be triggered. In this way the user can easily reset training to retract unpleasant warping. At 1000, S was kept small enough to allow intuitive interaction through fast convergence but large enough to generate meaningful results.

In case study 3 (fig. 6, fig. 7) the same mesh topology as in the previous model was employed to propagate information. But here the network nodes are represented by colored lines, each of which is described by the x, y, z position of their root, an x, y, z vector of their direction, their hue value and length. Neighbor lines are connected through quad mesh faces that incorporate color. Thus the simple mesh geometry of example 2 is extended to a grid structure with structural depth. This geometric interpretation exemplifies the potential of using SOM generated data to feed more complex parametric models over rather simple mesh geometries. For better usability the control objects are displayed as colored pipes and turn red, when they're being selected. From its

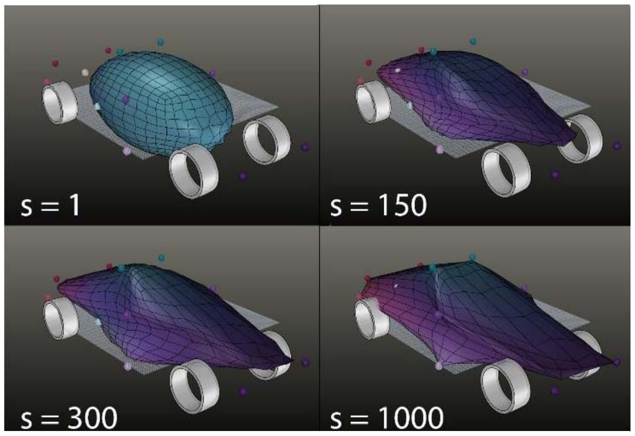

Figure 4: Case study 2, desktop display; 602 five-dimensional nodes connected with a quad mesh topology; S = 1000; convergence time: 300ms

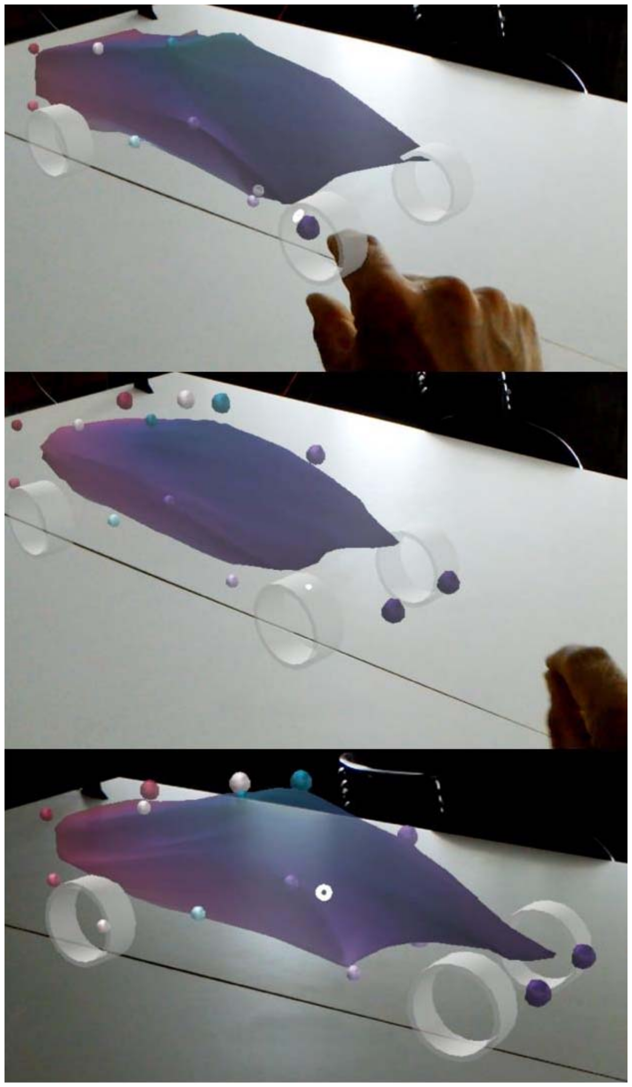

Figure 5: Case study 3: picking and dragging a control object to control the mesh geometry

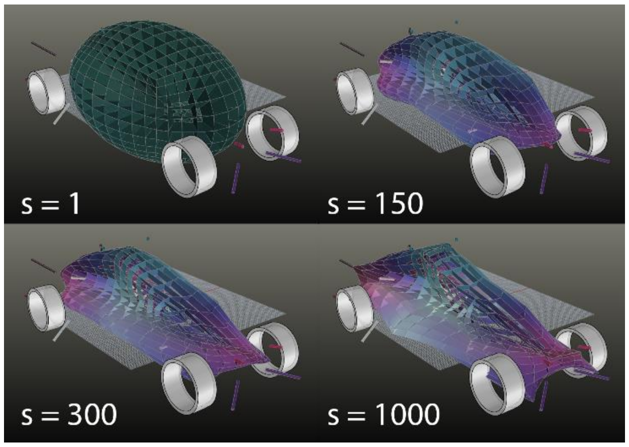

Figure 6: Case study 3; desktop display; 602 eight-dimensional nodes, connected with grid topology; S = 1000; convergence time: 300ms

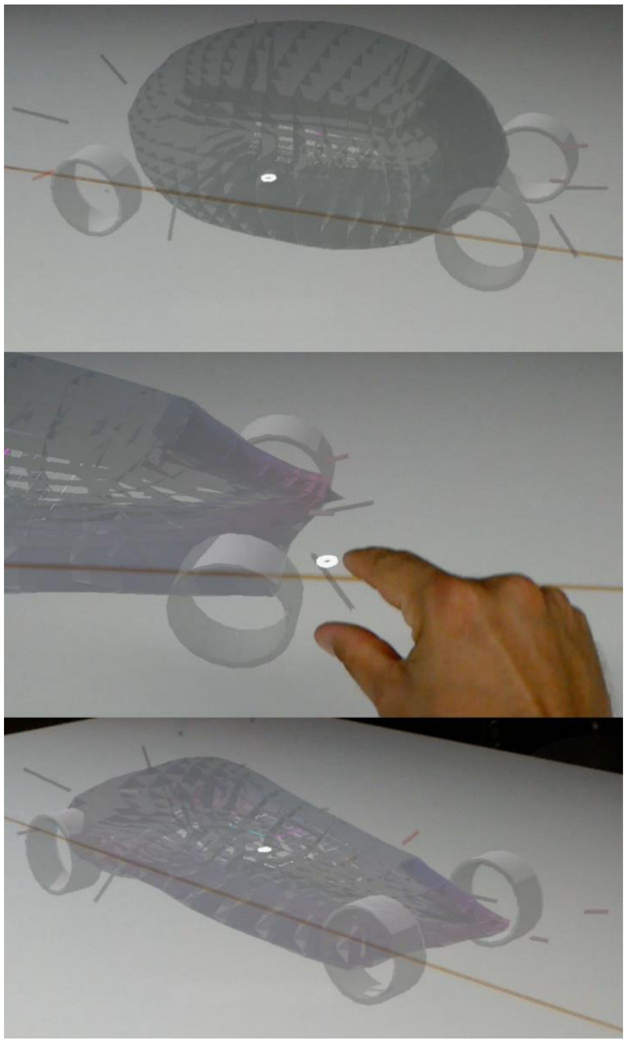

Figure 7: Case study 3: picking and dragging control objects to control mesh geometry

initial elliptic configuration the model converges almost as fast as in case study 2 when it is not transmitted to the Hololens. However, due to larger transmission data – the geometry consists of 1204 vertices and 2400 mesh faces – and more complex geometry to display the SOM computation was held for 10ms after each training cycle to avoid lag in the device synchronization. Along with increased cost for more complex mesh generation on the host side this resulted in a significantly longer convergence time of 16.7s. However, if the visualization of the network's training phase is not needed and only the final result is shown, a much faster interaction is possible.

VI. DISCUSSION

The discussed case studies serve to prove the general feasibility of modeling in mixed reality in conjunction with SOM-based geometric representations. However, problems of implementation in terms of speed and robustness must be addressed. In our setup we found the main bottle neck of speed to be the data transmission between host PC and the Hololens, resulting in a suboptimal responsiveness of the system. This issue could potentially be fixed through the use of a powerful portable host device in the form of a backpack and the connection through via USB. When a stationary host PC is preferred, a 5GHz Wifi connection or optimized transmission protocol could improve synchronization speed. Furthermore, due to the data's high dimensionality, the SOM model in case study three sometimes converged incompletely leaving visible gaps between control objects and closest nodes. Through a more fine grained learning procedure with increased S, this could be overcome at the cost of convergence speed. In addition, the geometry generation and display on the host side CAD environment made up a large portion of the computational cost. A more direct integration, without geometric visualization and a more efficient geometry generation outside Rhino3d and Grasshopper3d could further improve speed. Lastly the robustness in gesture recognition offered by the current model of Hololens could be improved further; major improvements in this regard in commercial products are foreseeable with the next release of Hololens.

ACKNOWLEDGMENT

Researchers on this project have received funding from the Collaborative Research Centre CRC 141 of the German Research Foundation.

REFERENCES

[1] T. P. Caudell; D. W. Mizell, "Augmented reality: an application of heads-up display technology to manual manufacturing processes." Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences. Ding, W. and Marchionini, G. 1992. A Study on Video Browsing Strategies. Technical Report. University of Maryland at College Park.

[2] H.-L. Chi, S.-C. Kang, X. Wang, Xiangyu "Research trends and opportunities of augmented reality applications in architecture, engineering, and construction" Automation in Construction. 33. 2013, 116–122. 10.1016/j.autcon.2012.12.017.

[3] D. Ren, T. Goldschwendt, Y. Chang and T. Höllerer, "Evaluating wide-field-of-view augmented reality with mixed reality simulation," 2016 IEEE Virtual Reality (VR), Greenville, SC, 2016, pp. 93-102. doi: 10.1109/VR.2016.7504692

[4] W. Lee, J. Park, "Augmented Foam: A Tangible Augmented Reality for Product Design" In: Proceedings of the 4th IEEE/ACM International Symposium on Mixed and Augmented Reality. Washington, DC, USA: IEEE Computer Society (ISMAR '05), 2005, pp. 106–109. Available online at http://dx.doi.org/10.1109/ISMAR.2005.16.

[5] F. Salim, H. Mulder, J. Burry, "Form fostering: A novel design approach for interacting with parametric models in the embodied virtuality", in Electronic Journal of Information Technology in Construction 16. 2011

[6] G. Quinn, A. Galeazzi, C. Gengnagel "Augmented and Virtual Reality Structures". IASS Annual Symposium, 2017: Interfaces: architecture . engineering . science At: Hamburg

[7] T. Kohonen, "Self-organized formation of topologically correct feature maps" In Biological Cybernetics 43 (1), 1982, pp. 59–69. DOI: 10.1007/BF00337288.

[8] Y. Liu, R.H. Weisberg, "A review of self-organizing map applications in meteorology and oceanography" In: Self-Organizing Maps-Applications and Novel Algorithm Design, 2011, p. 253-272.

[9] G. Zheng and V. Vaishnavi, "A Multidimensional Perceptual Map Approach to Project Prioritization and Selection," AIS Transactions on Human-Computer Interaction (3) 2, 2011, pp. 82-103

[10] J. C. Príncipe, R. Miikkulainen, S. Kiran, "Modeling the Bilingual Lexicon of an Individual Subject". Advances in Self-Organizing Maps. 2009, Berlin, Heidelberg: Springer Berlin Heidelberg.

[11] F. C. Moraes; S. C. Botelho; N. D. Filho; J. F. O. Gaya "Parallel High Dimensional Self Organizing Maps Using CUDA". 2012 Brazilian Robotics Symposium and Latin American Robotics Symposium. 2012 Brazilian Robotics Symposium and Latin American Robotics Symposium.

[12] O. Nechaeva, G. K. and Matsopoulos "Using Self Organizing Maps for 3D surface and volume adaptive mesh generation". In : Self-Organizing Maps. Rijeka: IntechOpen, 2010, pp. Ch. 9. Available online at https://doi.org/10.5772/9166.

[13] M. Zaghloul, "Machine-Learning aided architectural design". Self-Organizing map generate in between design alternatives. In Soddu C., Colabella, E. (eds.) Proceedings of the 18th Generative Art Conference, Venice 9, 10 and 11 December 2015

[14] J. Harding "Dimensionality reduction for parametric design exploration". In: Adriaenssens, S., Gramazio, F., Kohler, M., Menges, A. and Pauly, M., eds. (2016) Advances in Architectural Geometry 2016. Zurich, Switzerland: vdf Hochschulverlag AG an der ETH Zurich, 2016, pp. 274-287. ISBN 9783728137777

[15] T. Sederberg, S. Parry, "Free Form Deformation of Solid Geometric Models". ACM Computer Graphics, Proceedings of SIGGRAPH, S. 151–160, 1986

[16] M. S. Floater, "Mean value coordinates" In Computer Aided Geometric Design 20 (1), 2003, pp. 19–27. DOI: 10.1016/S0167-8396(03)00002-5.

[17] A. Jacobson, I. Baran, J. Popović, O. Sorkine: "Bounded Biharmonic Weights for Real-time Deformation" In ACM Trans. Graph. 30 (4), 2011, 78:1 ‐ 78:8. DOI: 10.1145/2010324.1964973.

[18] E. W. Weisstein, "Full Width at Half Maximum." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/FullWidthatHalfMaximum.html

[19] L. R. Herrmann, "Laplacian-isoparametric grid generation scheme", Journal of the Engineering Mechanics Division, 1976, 102 (5): 749–756.

[20] D. Davis, "Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture." PhD dissertation, RMIT University, 2013.