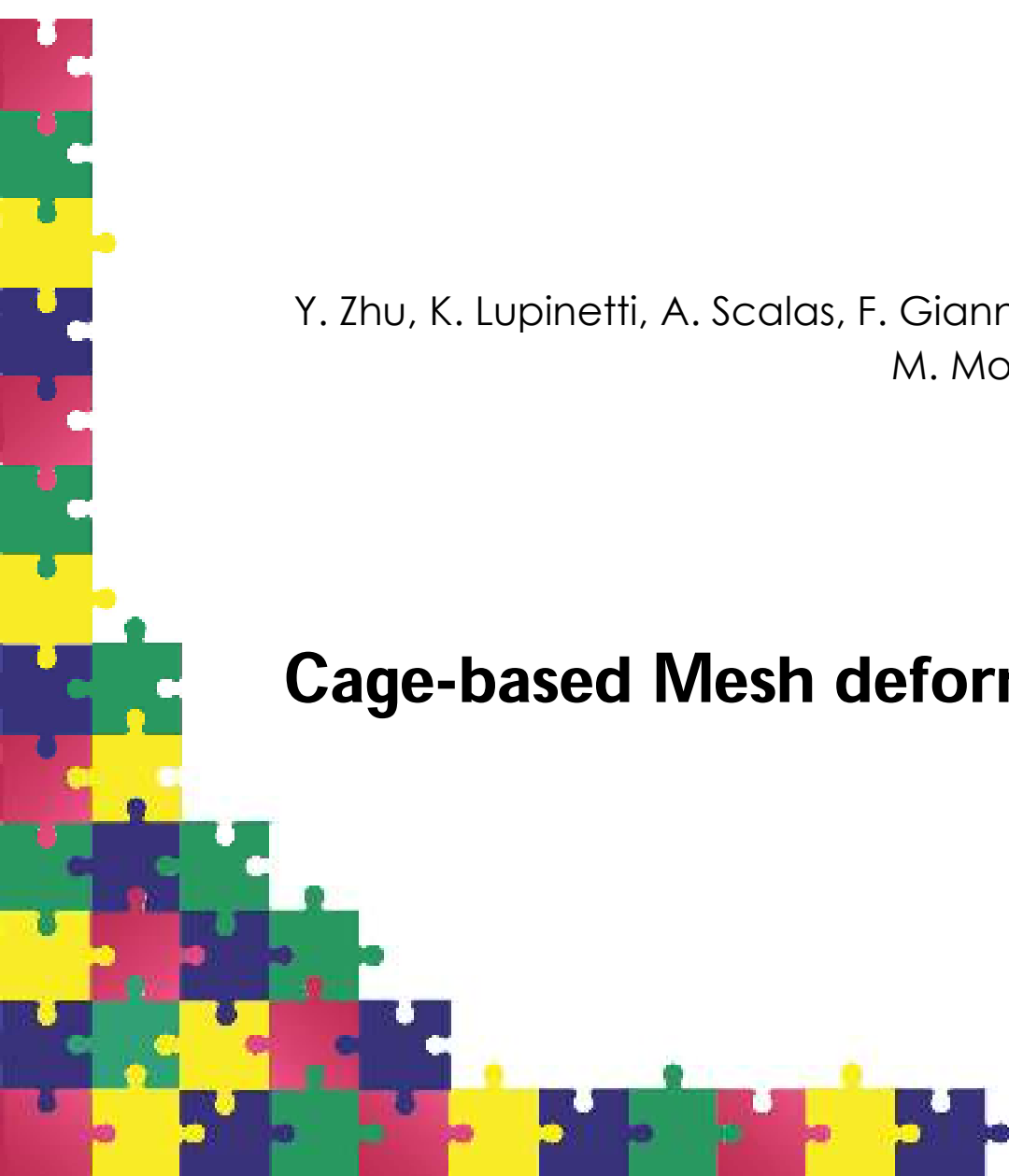


REPORT SERIES

A decorative graphic in the bottom-left corner consisting of a grid of colorful puzzle pieces (yellow, green, blue, red) arranged in a stepped, triangular shape.

Y. Zhu, K. Lupinetti, A. Scalas, F. Giannini, R. Lou, M. Monti,
M. Mortara, M. Spagnuolo

Cage-based Mesh deformation in VR

IMATI REPORT Series

Nr. 20-04

19 November, 2020

Managing Editor

Michela Spagnuolo

Editorial Office

Istituto di Matematica Applicata e Tecnologie Informatiche “E. Magenes”

Consiglio Nazionale delle Ricerche

Via Ferrata, 5/a

27100 PAVIA (Italy)

Email: reports@imati.cnr.it

<http://www.imati.cnr.it>

Follow this and additional works at: <http://www.imati.cnr.it/reports>

Cage-based Mesh deformation in VR

Yuanjou Zhu, Katia Lupinetti, Andreas Scalas, Franca Giannini, Ruding Lou,
Marina Monti, Michela Mortara, Michela Spagnuolo



Yuanju Zhu

Arts et Métiers

Campus de Cluny, Chalon-sur-Saône, France

E-mail: yuanju.zhu@ensam.eu

Katia Lupinetti

Istituto di Matematica Applicata e Tecnologie Informatiche "E. Magenes"

Consiglio Nazionale delle Ricerche

Via de Marini, 6 (Torre di Francia) - 16149 Genova, Italy

E-mail: katia.lupinetti@ge.imati.cnr.it

Andreas Scalas

Istituto di Matematica Applicata e Tecnologie Informatiche "E. Magenes"

Consiglio Nazionale delle Ricerche

Via de Marini, 6 (Torre di Francia) - 16149 Genova, Italy

E-mail: andreas.scalas@ge.imati.cnr.it

Franca Giannini

Istituto di Matematica Applicata e Tecnologie Informatiche "E. Magenes"

Consiglio Nazionale delle Ricerche

Via de Marini, 6 (Torre di Francia) - 16149 Genova, Italy

E-mail: franca.giannini@ge.imati.cnr.it

Ruding Lou

Arts et Métiers

Institute Image - Laboratoire d'Ingénierie des Systèmes Physiques et Numériques (LISPEN)

Campus de Cluny, Chalon-sur-Saône, France

E-mail: ruding.lou@ensam.eu

Marina Monti

Istituto di Matematica Applicata e Tecnologie Informatiche "E. Magenes"

Consiglio Nazionale delle Ricerche

Via de Marini, 6 (Torre di Francia) - 16149 Genova, Italy

E-mail: marina.monti@ge.imati.cnr.it

Michela Mortara

Istituto di Matematica Applicata e Tecnologie Informatiche "E. Magenes"

Consiglio Nazionale delle Ricerche

Via de Marini, 6 (Torre di Francia) - 16149 Genova, Italy

E-mail: michela.mortara@ge.imati.cnr.it

Michela Spagnuolo

Istituto di Matematica Applicata e Tecnologie Informatiche "E. Magenes"

Consiglio Nazionale delle Ricerche

Via de Marini, 6 (Torre di Francia) - 16149 Genova, Italy

E-mail: michela.spagnuolo@ge.imati.cnr.it

Abstract.

This report describes the integration of cage-based mesh deformation into in a VR application. The application allows multimodal interaction with the shape at different level of semantic information content. According to the requirements, users can perform some mesh deformation interactively acting on single or multiple control points possibly corresponding to semantically meaningful annotated object subparts, which can be simultaneously selected. The system takes as input an annotated mesh and its control cage. Shape annotations can be hierarchical: a mesh sub-region can simultaneously refer to several annotations, referring to different levels of specifications.

Keywords: *3D shape deformation, VR, multimodal interfaces, natural interface*

Contents

1. Work objectives	1
2. Background and selected technologies	1
2.1. Cage-based mesh deformation.....	1
2.2. Controllers for gesture detection	3
2.3. Annotations.....	8
2.4. Related work	9
3. The developed system	10
3.1. The imported data	11
3.1.1 The barycentric coordinates	11
3.1.2. Annotations	11
3.1.3. Computation of the association between annotated segments and cage vertices	12
3.1.4. The creation of the handles for the deformation	13
3.2. The developed functionalities.....	14
3.2.1. The visualization and interaction with the annotated model	14
3.2.2 Control point selection	17
3.2.2.1. Mouse version	17
3.2.1.2. Leap Motion version	17
3.2.3. The translation of the control points	21
3.2.3.1. Mouse version	21
3.2.3.2. Leap Motion version	23
3.2.4 Rotation	23
3.2.4.1. Mouse version	23
3.2.5. Scale the model	24
4. Conclusion and future work	26
References	27

1. Work objectives

The advances in Virtual and Augmented Reality technologies, including the improvements in the capabilities to trace human movements and gestures at low cost (easily affordable by small companies, professionals and even by amateurs) pave the way to the development of more natural shape interaction and modification applications usable in various industrial and leisure contexts. For instance, it opens the possibility for companies to include end-users in product evaluation and customization. Other industries that strongly benefit from these new technologies are those addressing training, marketing, culture and entertainment. In these domains, game engines are normally used to create interactive applications. In all these contexts, 3D models are normally created in dedicated authoring tools, such as Computer-Aided Systems, and require further adaptations to be exploited in the VR environment. Therefore, any change of the 3D shape requires to loop back to the authoring tool and to re-perform the various preparation steps. To overcome this time consuming loop, shape modification capabilities should be provided in the VR environment.

To this aim, the purpose of this work is the development of a virtual reality application allowing cage-based mesh deformation with multimodal interaction, i.e. with gaze, voice commands and bare hand gestures. Deformations are performed by the manipulation of several handles that can be single and/or groups of control points. Groups of control points corresponding to meaningful constituting object sub-parts can be simultaneously selected thanks to the annotations provided over the 3D mesh.

The rest of the document is organized as follows: Section 2 provides a short overview of related concepts while describing the characteristics and information provided by the controller adopted for the gesture capture; Section 3 describes the developed prototype and Section 4 provides some conclusions and indications of future work.

2. Background and selected technologies

2.1. Cage-based mesh deformation

As a geometric model interactive editing technology, mesh deformation has important value in geometric modeling and computer animation. There are different techniques for mesh deformation: Free-form deformation (FFD), skinning, Radial Basis Functions (RBF), curve based deformation and physics simulation [NS12].

In this work we exploit cage-based deformation which is a skinning technique gaining interest in recent times due to its efficiency toward real time deformation since it is computed by simple matrix products.

Given a mesh, a cage is defined as any closed mesh that envelops the mesh itself. The cage is usually a simplified version of the mesh and contains much less vertices; it finds several applications, such as collision detection and in general everywhere a coarser version of a mesh could be useful e.g., for reducing the computational complexity as in the case of deformation. In cage-based deformation techniques, vertices of the cages are used as control points, such that changes applied to cage coordinates modify the mesh accordingly. In particular, cage-based deformation techniques base on the concept of Generalized Barycentric Coordinates (GBC), which give means for defining the value of a certain function over a point in space (and so, for example, over a mesh vertex) in terms of a linear combination of some control points (cage vertices).

The general formula for the computation of the value of a function f over a vertex v_i of the mesh model is

$$f(v_i) = \frac{\sum_{j=0}^n w_{ij} f(c_j)}{\sum_{j=0}^n w_{ij}} \quad (1)$$

Where c_j is the j -th vertex of the cage and w_{ij} is the weight associated with v_i and c_j . So, if we take $f(x) = x$, we can use the previous formula for computing the position of the mesh vertices using the positions of the cage vertices. The GBC are homogeneous and need to be normalized by

$$\lambda_{ij} = \frac{w_{ij}}{\sum_k w_{ik}} \quad (2)$$

The same result may be obtained by applying the corresponding formula in matrix form:

$$M = B \cdot C \quad (3)$$

Where M is a matrix reporting the position in space of the mesh vertices, B is the weights' matrix and C is the matrix reporting the position in space of the cage vertices.

This method allows to retrieve the position of a mesh vertex from the position of the cage vertices and. depending on the associated weight (with value between 0 and 1). When the weight equals 0, the associated mesh vertex is unaffected by the corresponding vertex on the cage. When the weight equals 1, the cage vertex completely controls the motion of the mesh vertex, which is fully affected by the vertex of the cage. Weights within the range allow multiple cage vertices to affect a vertex of the model.

2.2. Controllers for gesture detection

Gesturing is becoming more and more important in AR/VR. There are many hand tracking devices with gesture recognition methods on the market, such as Microsoft's Kinect, Google's Project Soli; Dexta Robotics's data gloves; Ascension's trakSTAR, Leap Motion controller, etc.

Kinect uses infrared rays that are invisible to the human eye. The laser light is evenly distributed and projected into the measurement space through the diffuser in front of the lens, and then each speckle in the space is recorded by an infrared camera. After capturing the original data, an image with 3D depth can be created. It can be used in color recognition, distance detection, and skeletal tracking.

The Project Soli (figure 1) radar chip uses a miniature radar to capture subtle gestures [Soli]. The radar mainly uses the reflection of radio waves for imaging. The computer can get the position and the speed of the object, and the movement of the object can be captured finely by comparing the transmitted wave and the reflected wave.

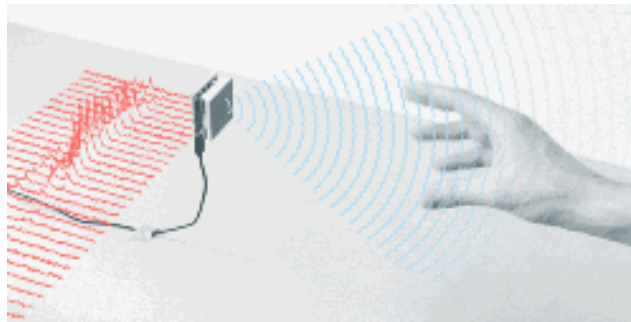


Figure 1: Use Soli to detect hand gestures

Data gloves (figure 2) are special gloves with a built-in sensor on. The sensor detects the flexion and extension angle or position of the fingers, and then we can get the position of the hand based on inverse kinematics. Data gloves are very accurate for the detection of local hand movements and are not restricted by the visual field in the vision. However, it is inconvenient to wear gloves on the hand, and it can only detect partial finger movements and needs additional trackers to locate the overall position and angle of the hand.



Figure 2: Dexta Robotics company's data gloves(DEXMO)

The trakSTAR (figure 3) requires 6 magnetic sensors on the hands and a magnetic transmitter in front of them [Trak]. The magnetic transmitter will create an electromagnetic field within a certain range, and then infer the position angle of the fingertip and the palm according to the different electromagnetic field intensity detected by the sensor at different positions and angles in the electromagnetic field. Then by using inverse kinematics, we can determine the position of all hand joints.

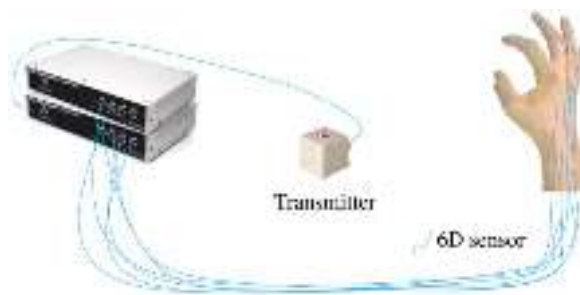


Figure 3: Schematic diagram of how to use trakSTAR

The Leap Motion is a compact sensor that exploits two CMOS cameras capturing images with a frame rate of 50 up to 200fps. It is very suitable for hand gesture recognition because it is explicitly targeted to hand and finger tracking. In addition, it can be mounted on the headset or embedded in desktop interfaces, and software pipelines able to provide hand pose estimation from RGB and RGBD data are available. In this work, Leap Motion has been selected as input device due to its easy availability, affordability and transparent integration with desktop and immersive VR applications. Therefore, the

following section provides some information on the Leap Motion sensor and on its usage for gesture understanding useful for the implementation of the described application.

2.2.1. The leap motion controller

The Leap Motion Controller is very convenient to use and portable with a small dimension of 0.5x1.2x3 inches (see figure 4). To use the Leap Motion Controller, the user only needs to install the leap controller SDK and connect the controller to a computer or to an Head-Mounted Display (HMD) by USB. Then the user put the hands on top (or in front) of the Leap Motion Controller.

There are three Infrared Light emitters and two cameras receiving the IR lights.

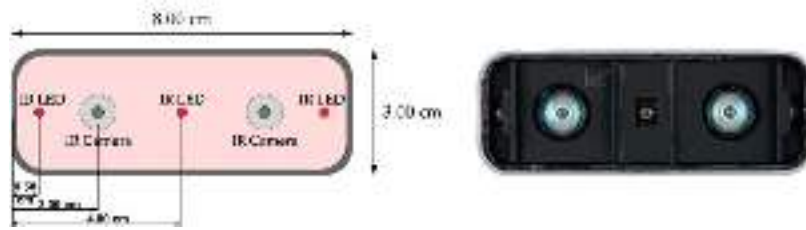


Figure 4: The leap motion controller's structure

The Leap Motion Controller can capture hand movements by tracking specific hand nodes called joints. For each joint, the Leap Motion sensor collects information on position, direction and velocity. More precisely, the Leap Motion sensor collects the following data:

1. *palm_normal* (vector): The normal vector to the palm (see figure 5a).
2. *palm_position* (vector): The central position of the palm in millimeters in the coordinate system centred in the Leap Motion Controller origin.
3. *palm_velocity* (vector): The rate of change of the palm position in millimeters/second.
4. *grab_strength* (float): the strength of a grab hand pose as a value in the range [0...1]. An open hand has a grab strength of zero. As a hand closes into a fist, its grab strength increases to one.
5. *finger[i].tipPosition* (vector): the instantaneous position of a finger in the coordinate system centred in the Leap Motion origin. (see figure 5b, $i = [0,4]$, representing thumb, index, middle, ring and pinky respectively.)
6. *finger[i].direction* (vector): the current pointing direction vector of a finger. ($i = [0,4]$, representing thumb, index, middle, ring and pinky respectively.)



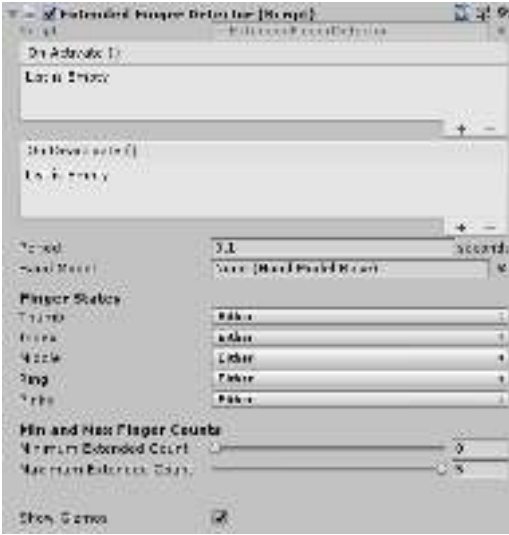
Figure 5: (a) palm_normal is expressed as a unit vector pointing in the same direction as the palm normal (a vector orthogonal to the palm); (b) finger.tipPosition and direction vectors provide the positions of the fingertips and the directions in which the fingers are pointing.

The Leap Motion Controller provides a detection accuracy of about 200 μm . The Orion or higher versions of Leap Motion Controller do not provide built in gesture recognition functionalities.

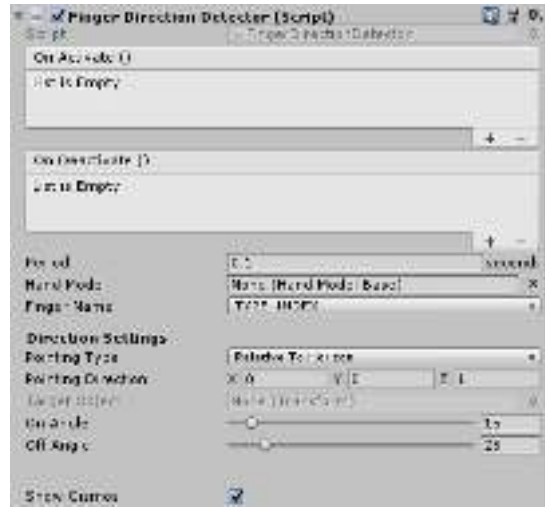
With Leap Motion it is possible to recognize both static and dynamic gestures. Static gestures can be identified by analyzing the relative distances between palm and fingers. Dynamic gestures can be identified considering also the velocity magnitude of the joints (in addition to the joint positions). The Leap Motion system provides a set of scripts that can be used for the recognition of certain gestures. Here are some detectors that might be of interest for the device exploitation:

- *Extended Finger Detector* (figure 6a) A detector that allows detecting if a finger is extended or not.
- *Finger Direction Detector* (figure 6b) determines if a finger is pointing to a certain direction (user-defined, can be relative to camera, horizon, world or a target).
- *Palm Direction Detector* Similar usage as “Finger Direction Detector”.
- *Detector Logic Gate* (figure 6c)

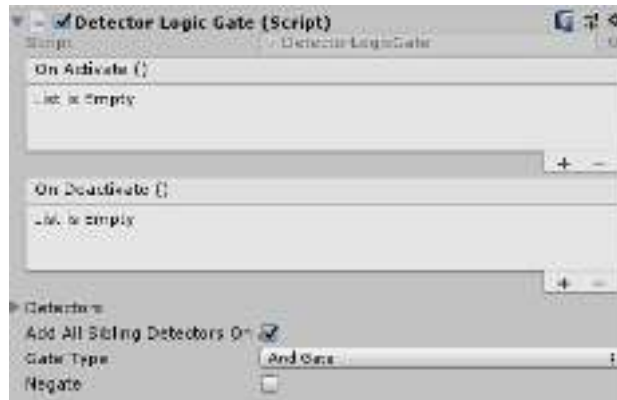
When designing complex gestures, the “Detector Logic Gate” allows combining detectors with logic, e.g., a “Extended Finger Detector” can be used to detect if all the left fingers are extended and a “Finger Direction Detector” allows to detect if the right index finger is pointing to a target. The use of “and” in the “Logic Gate” allows to link the “Finger Direction Detector” and “Extended Finger Detector”, so that the event in the logic gate can be triggered when the left hand fingers are extended and the index finger of the right hand is pointing to a target.



a)



(b)



(c)

Figure 6: A subset of LM detectors: (a) Extended Finger Detector; (b) Finger Direction Detector; (c) Detector Logic Gate.

When the Infrared Radiation (IR) cameras cannot receive enough information to reconstruct the gesture by using the leap hands (e.g. when the fingers are self-occluded by the palm), the Leap Motion Controller will predict a gesture which is often not correct. So the user should try to keep hands visible for the leap motion and avoid important fingers or regions to be self-occluded by other hand parts to have good quality data.

In order to use the leap motion controller, *aleap hands prefab* must be configured. According to the version of the core asset, different prefabs are provided, it is also possible to design own hand prefabs. In this project, we include the core asset Orion_4.1.5 using the hand prefab named “Leap Rig” (available in the leap motion core assets file). It consists of the following objects:

1. The *Leap Rig* is the root object. This is the transform to manipulate for moving the player in the XR space. This object contains the optional *XRHeightOffset* script by default.
2. The *Main Camera* is a direct child of the Leap Rig. Its local position and rotation are controlled directly by Unity's XR integration, so it cannot be moved manually (its parent, the Leap Rig, has to be moved). In the project, we did not use the Main Camera but replaced it with our FPS camera and we attached the component named “Leap XR Service Provider” to the FPS camera. It retrieves hand data from the sensor and also accounts for latency differences between the sensor and the headset's pose tracking.
3. *Hand models* is a game object that contains all the leap hand models we may use. In this work, we used the “Capsule Hand Left” and “Capsule Hand Right”.
4. The *Hand Model Manager* can be found in the interaction engine file. We drag it under the “Leap Rig” object so it is a sibling of the FPS camera and is the parent object for all of the HandModels, such as Capsule Hands and Rigged Hands, etc. In this work only “Interaction Hand (Left)” and “Interaction Hand (Right)” were considered.
5. To add some attachments to the hands, the “*Attachment Hands*” in the core assets file has to be used and dragged into the “*Leap Rig*” game object. It allows to add buttons or sliders to the attachments on the leap hand to trigger some events.

2.3. Annotations

The term annotation refers here to a mechanism that links a sub-portion of the geometrical representation of an object to some related information [SMS20]. The purpose of annotation is to create correspondences between objects, or segments, and conceptual tags [RASFO7]. The possibility to operate on selected semantically meaningful areas could provide faster algorithms, better results and more pleasant interfaces. Just to give a few examples, the search for a specific shape could be a lot simpler introducing a filter to visualise only the shapes resembling a specific object or containing a specific part (e.g., search for “teapot” shapes containing a “flower” pattern); visualisation interfaces can be defined for highlighting certain parts (e.g., focus on the shape of the “pump” in a fridge assembly shape); and, of course, manipulation interfaces could greatly benefit from the introduction of semantic annotations.

In this work, we assume the object has already been annotated and we import the “semantised” shape into our system by loading geometry and tags from a structured file.

2.4. Related work

During the last decade, the interest in the development of immersive VR and AR systems for manipulating and modelling 3D environments through gestures has increased [CGM19,MCG19]. Most of the Head Mounted Displays are also providing some tools for shape modelling. These commercial systems either address the generation of objects for games and animation or try to integrate the CAD user interface in VR environments using controllers replacing desktop mouse providing buttons and pointing capabilities. Shape sculpting by push and pull operations together with the combination of primitives are the main capabilities of commercial applications mainly devoted to the animation domain [Lea, Ocu, Mak]. Conversely, surface control point editing is the main capability provided for tools addressing the industrial design domain [Min, Gra], using the controllers acting as a 3D mouse. Mesh deformation has always been a hot topic for researchers, especially in recent years with the popularization of virtual reality technology [MCGF19]. As an example, John Akers [Ake] has developed “TryDesVR” applying mesh deformation in virtual reality, users can have a more intuitive understanding of 3D models (especially scale and depth) by using it. The “TryDesVR” allows users to interact and modify the model before having it 3D printed. Skeleton-based shape deformation is applied in TRyDesVR.



Figure 7: Before(a) and after(b) images of mesh deformation using generated handles, extracted from [Ake].

Another example is the work done to perform interactive virtual sculpting using camera-based hand tracking [GWF18], see fig. 8. Vinayak and Ramani [VR15] developed a system for modelling and shaping pottery objects using Leap Motion Controller. In their work, they address the problem of determining how the shape and motion of a user’s hand and fingers geometrically relates to the user’s intent of deforming a shape. Thus, they consider that the expression of user intent in the shaping processes can be derived from the geometry of contact between the hand and the manipulated object. But these works only support rotational symmetry in a pottery context.



Figure 8: The UI in DigiClay (Large Picture) and user interaction (Small Picture).

The GoogleVR application “Blocks” allows users to create low-polygon models with a focus on constructing models from primitive shapes, but it is not designed to support complex meshes.

Anyhow, to our knowledge most of the works present in literature are not exploiting the semantic segmentation of the object to perform rather localized and more semantic oriented shape modifications.

3. The developed system

Figure 9 provides an overview of the developed system exploiting the Unity 3D graphical engine easily integrated with HMD and leap motion controllers.

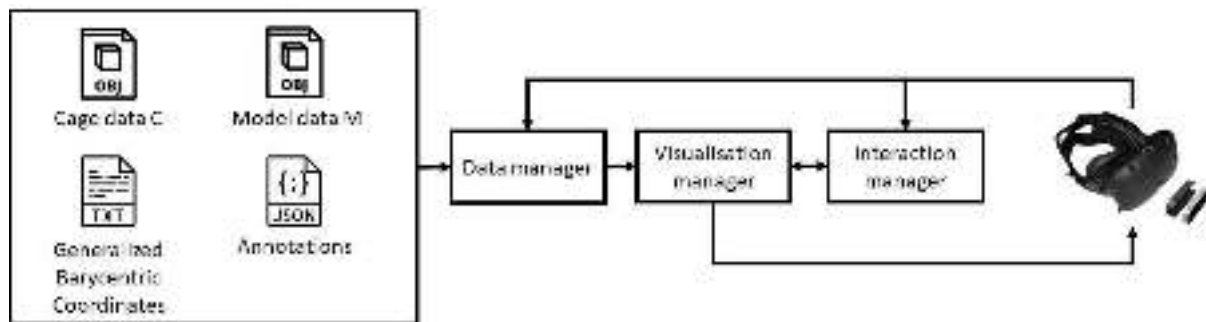


Figure 9: The overview of the developed system

The system takes in input various information, namely: a triangular mesh model M , the cage mesh C , the Generalised Barycentric coordinates and the annotations associated to the mesh subparts. This information is stored in different files which are further processed to create the suitable data structure within the Unity 3D environment.

3.1. The imported data

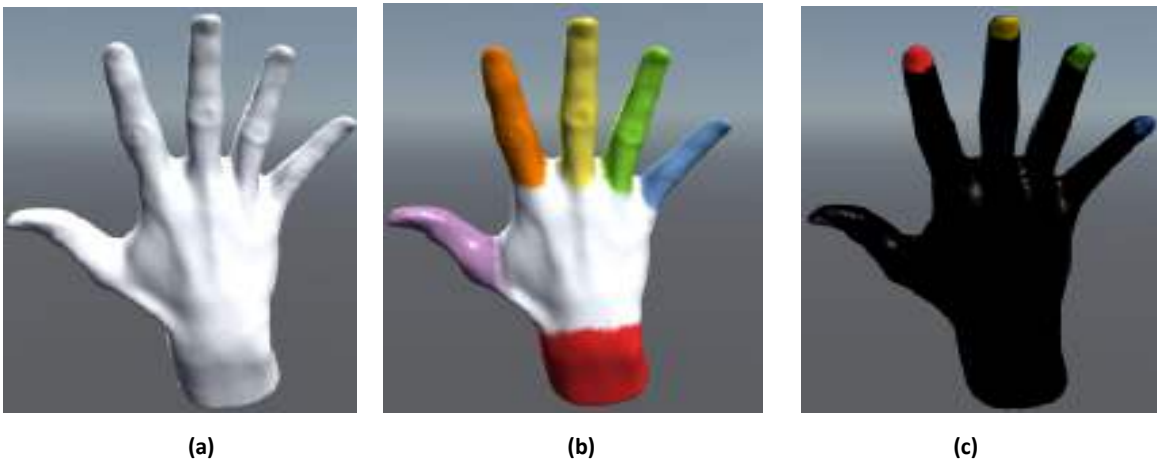
3.1.1 The barycentric coordinates

Mean value coordinates (MVC) are good solutions for value interpolation, and their simple closed form expressions encourage their usage in the fastest real-time applications. But the fact of having negative values and global deformation makes this method less interesting to character articulation. We aim at addressing general shapes, i.e. objects of interest in industrial design, which do not have character articulation, so we adopted MVC, which are assumed to be a priori given. Anyhow, the developed architecture and functionalities do not depend on the adopted type of barycentric coordinates, therefore changing coordinates is transparent to the system.

3.1.2. Annotations

We assume parts are annotated and that annotations correspond to meaningful parts of the object which can be modified as a whole, thus corresponding to Regions of Interest (annotated RoIs) for the shape modification. They can overlap (e.g., *nail* overlap with the *finger* part of the *hand*) hence indicating a hierarchy among sub-parts. Therefore, we arrange parts into a tree, having the whole object annotated with the name of its class (e.g., “hand”), as the root. Starting from the root node, we build the hierarchy of parts populating the tree, checking the containment of annotations – an annotation is contained inside another if all its associated triangles belong to the other one as well.

Figure 10 shows an annotated hand and the corresponding hierarchy of parts. Therefore, each annotated mesh subpart is stored as a tree node instance characterized by the attributes indicated in fig. 11.



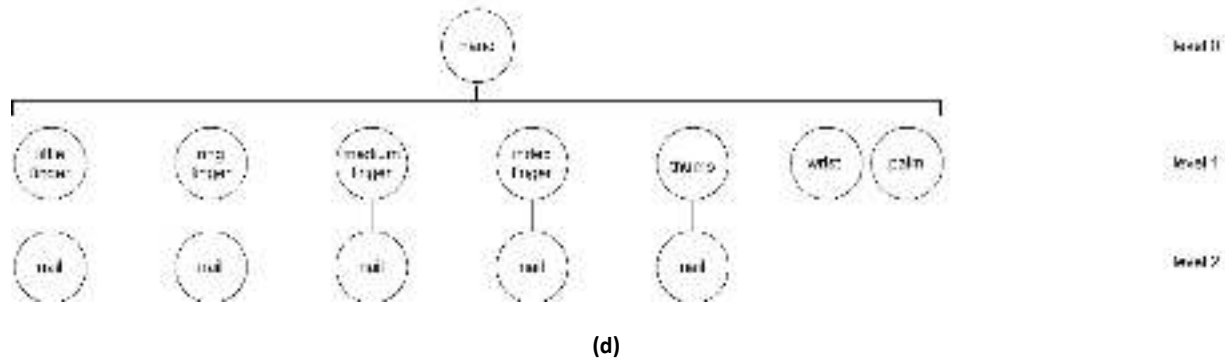


Figure 10: Example of annotation hierarchy tree (d) and related shape segments: (a) level 0; (b) level 1; (c) level 2

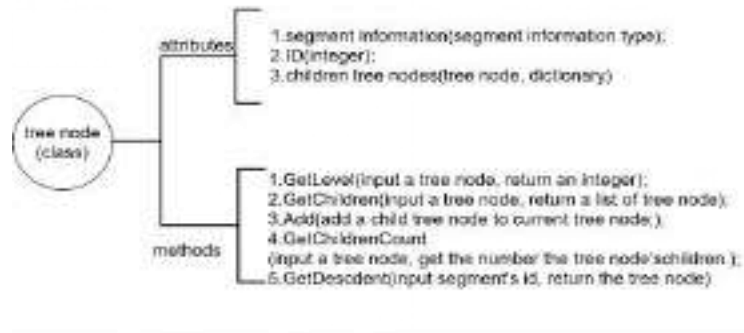


Figure 11: the tree node class adopted

3.1.3. Computation of the association between annotated segments and cage vertices

To modify the shape, we operate on the cage vertices. Any modification of the cage vertices will be used for updating the position of the vertices of the model by computing the matrix multiplication $B \cdot C$. To have the possibility of selecting all the cage vertices associated to specific annotated sub-regions, it is necessary to understand which cage points have major influence on each annotated region.

Once the vertices of the model associated to a certain segment have been identified, we use their indices to understand which rows of the barycentric coordinates matrix correspond to them. Then, we check which values in these rows are bigger than a certain threshold (within 0 and 1) to figure out which cage vertices have the most influence on the segment.

Some examples of association between cage vertices and sub-parts can be seen in figure 12.

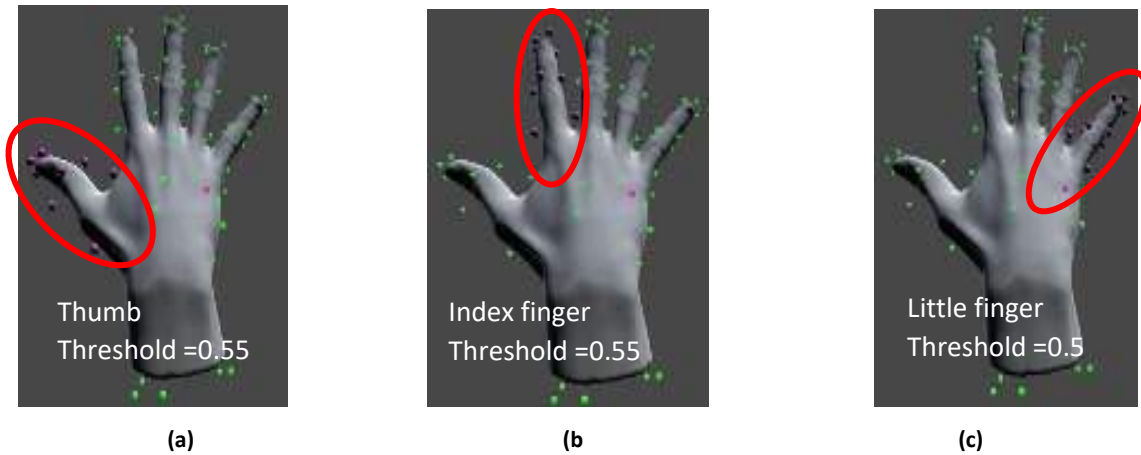


Figure 12: The examples of segmentation of the cage's vertices: purple spheres are the generated at the vertex of the cage if the vertices have an influence bigger than the threshold on the vertices of the corresponding segment on the model (a) thumb with a threshold = 0.55, (b) index finger with a threshold = 0.55, (c) pinky with a threshold = 0.5)

Clearly the best threshold covering all the cage vertices-segment association depends on the specific object, for instance for the hand model, it has been found that with the threshold = 0.4, the vertices on the cage can cover correctly the whole segment on the model for all the segments.

3.1.4. The creation of the handles for the deformation

In this work, handles are single or sets of cage nodes, which are then moved by the user to achieve the wished deformation. To be selected and moved they need to belong to the VR environment and this is possible creating “game objects” that represent the handles in the Unity 3D environment such that the user can interact with them. Control points are visualised as spheres, whose color changes when selected or gazed at. The default color is given by the blending of the colors of the different annotated regions associated to the corresponding cage vertex. Cage vertices are stored in a matrix (figure 13 a), where every row in the matrix is the 3D position of a control point on the imported file (figure 13b).

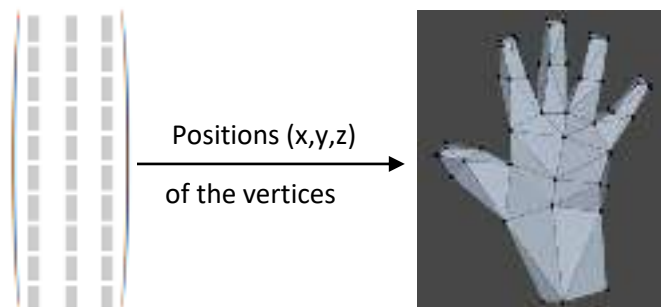


Figure 13: (a) The matrix of cage (C); (b) The cage of the model with control points (in black)

When two control points are selected (see figure 14), they can be moved (see figure 15a) and the position of the vertices of the cage mesh is updated (see figure 15c).



Figure 14:(a) The matrix of the cage elements (the two rows represent the two selected control points);
(b) the cage mesh with two control points selected

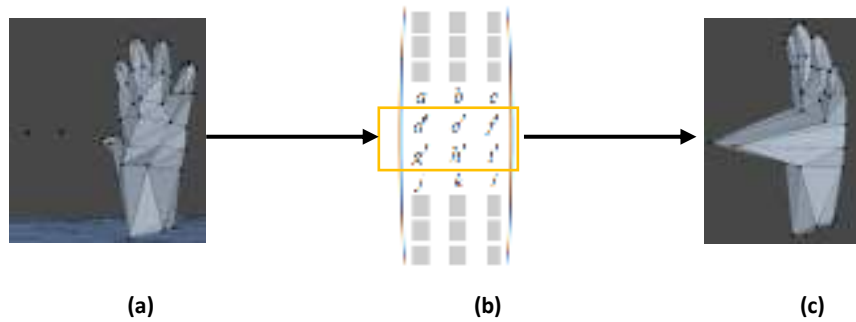


Figure 15: (a) The position modification of two control points; (b) the matrix of the cage (the highlight part represent the moved “control points”); (c) the update of the cage mesh

3.2. The developed functionalities

The cage-based deformation has been developed allowing either mouse or leap motion controller for the user-system interaction. The mouse version has been developed for verifying the data structure and functionalities before developing the interaction with bare-hands supported by the leap motion sensor. Thus, the underlying functionalities are the same, while the user interaction is adapted to the interaction modality.

3.2.1. The visualization and interaction with the annotated model

As previously explained, in our work we intend to exploit annotations over the 3D model to ease the selection of groups of control points and to perform localized modifications over semantically meaningful regions. Thus, the first functionality to be provided is the capability to visualize how the

object has been annotated either in terms of semantic mesh segments and of their hierarchical organization and associated labels. To visualize the model at different semantic hierarchy levels, the work can be divided into two aspects: visualization of the mesh segments and visualization of their related control points.

Since mesh elements can belong to several annotated regions, it is particularly important to communicate such information. To this aim, we decided to define color blending for their rendering. Thus, for each annotated segment at a given hierarchy level a color is assigned both to the mesh element and to the associated control points. To provide the information of the membership to multiple annotated segments, we render each triangle and each control point with a color corresponding to the blending of that of the various segments to which they belong to. More specifically, we fill in a color array by using the segments' color and vertex indexes of the model (the steps 1-4 in the figure 16 below) and mix the colors (compute the average RGB values of the colors and create a new color with these RGB value) if more than one color is filled in the array.

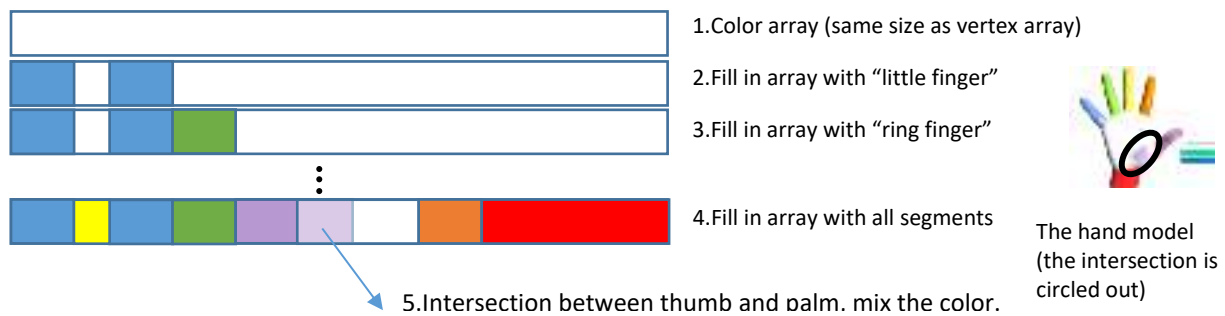


Figure 16: mixing the associated segments' color

Figure 17 shows the visualization of the various annotation hierarchy levels associated to the hand model: at level 0 there is only one segment corresponding to the whole hand (figure 17.a.); at level 1 (figure 17.b) there are 7 segments (5 fingers, palm and wrist), where the thumb and the palm have an intersection region thus the shared region is colored by the average between the color of the two segments; at level 2 (figure 17c) there are 5 segments (5 nails), the remaining part does not belong to a segment and its color is black.

The control points displayed at different levels have the same color as their associated model segment, an example of the control point at different levels is illustrated in figure 17. Figure 17a shows the set of control points corresponding to the entire set available control points, then acting on that selection the modification is propagated to the entire model. In figure 17b, there are 7 sets

of control points corresponding to the 7 segments. The control points that strongly influence more than one segment display a color obtained by the blending of the color of the associated segments (e.g., the control points that affect palm and wrist have a “pink” color, which is a blending of white and red). Finally, in figure 17c, there are 5 sets of control points that belong to the 5 nail segments, while the other control points have no color because they belong to parts on the model that are not annotated.

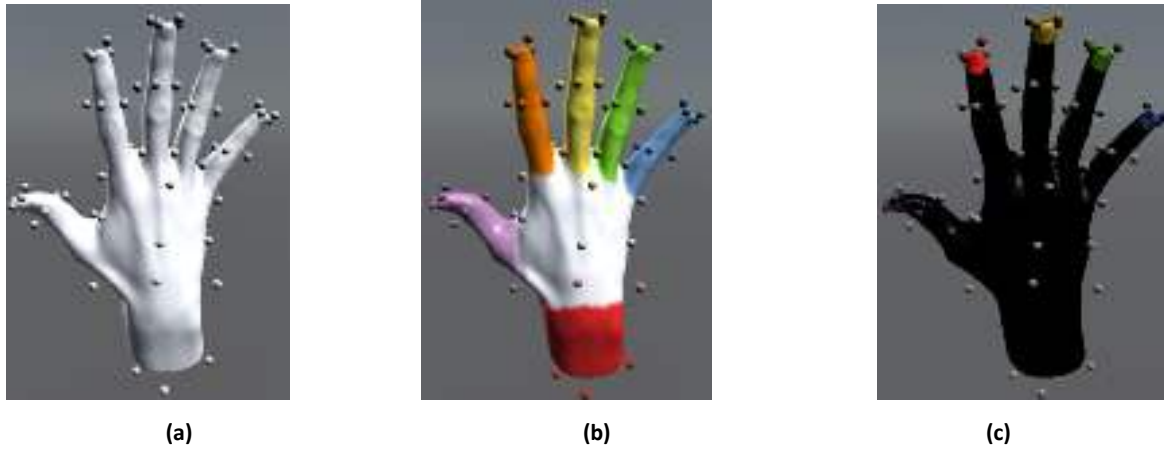


Figure 17: The control points displayed at different annotation hierarchy levels (a) level 0; (b) level 1; (c) level 2)

To communicate the tags associated with the various segments, we create 3D texts at the barycenter of each segment, where they will rotate to always face the user. When the level is changed, the previous tags will be destroyed and we create new tags for the current level (see figure 19).

The way to browse along the annotation hierarchy and select a specific level depends on the adopted interaction mean as in the following sections.

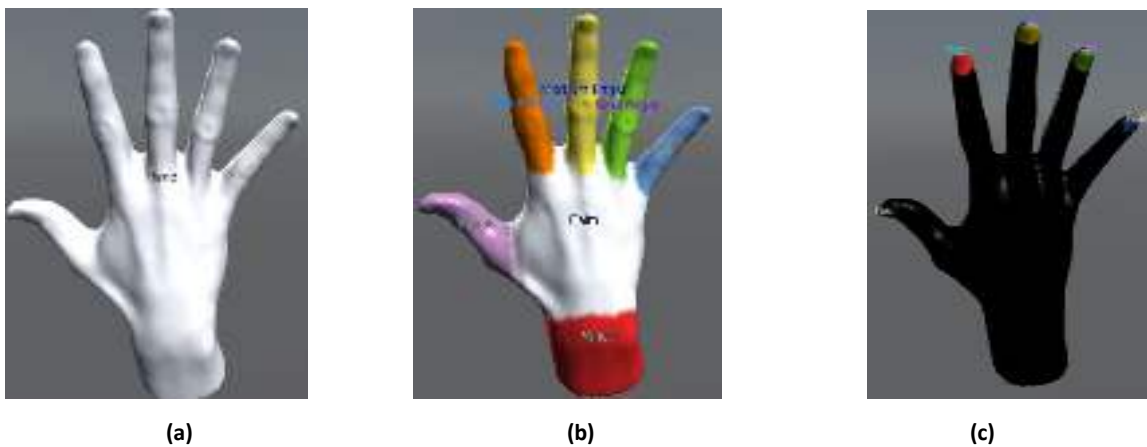


Figure 18: The tags of the segments displayed on the model (a) level 0; (b) level 1; (c) level 2

3.2.1.1. Mouse version

In the mouse version, the level is chosen by voice command specifying the desired level within the available level range. If the level is changed there will be a feed at the “voice command” text box.

3.2.1.2. Leap motion version

For browsing within the levels, the “swipe hand upwards” or “swipe hand downwards” can be used after the “change level” mode activation obtained by saying the corresponding voice command, obtaining an effect coherent with the performed action (“swipe hand upwards” will increment the level of the hierarchy to be visualized by 1 and “swipe hand downwards” will do the opposite). Alternatively, the same voice commands of the mouse version are available.

3.2.2 Control point selection

When the user imports a model, the object at level 0 is shown together with the information of the range of the available annotation levels, but no control points are shown. To generate the control points so that the model can be modified, the user has to select a level.

3.2.2.1. Mouse version

When the user moves the mouse and left-clicks on a control point, the control point is selected, the index of the control point is stored in the so-called *selection list* and its color is changed to red.

To move the selected control points simultaneously within Unity 3D environment, the control points in the *selection list* are placed in one parent game object (“*selected control points*”).

To deselect a control point, the user has to move the cursor to a selected control point and right click; in this way, the control point is removed from the *selected control points* game object and from the *selection list* and recovers its original color.

To allow the user to select several control points at one time, we apply a standard metaphor adopted in selecting elements in files, i.e. by pressing the shift key in the keyboard while left clicking the mouse and moving it to design a rectangle which contains the control points to be selected, i.e., the control points that are within the rectangle defined by the starting and ending position of the mouse will be selected. The opposite applies by clicking the right button.

3.2.1.2. Leap Motion version

For the second version, we considered using a combination of gaze, voice commands and Leap Motion Controller for the selection of the control points. Then, the control points are moved according to a series of predefined gestures, while the actual deformation on the mesh is propagated

in the same way of the mouse version. Various metaphors have been evaluated for the control point selection and are briefly described in the following.

The first metaphor considers the selection by touching the control point game object with a finger of one hand, e.g. the left one, while the use of the other hand indicates the deselection of the touched point, see figure 19 below:

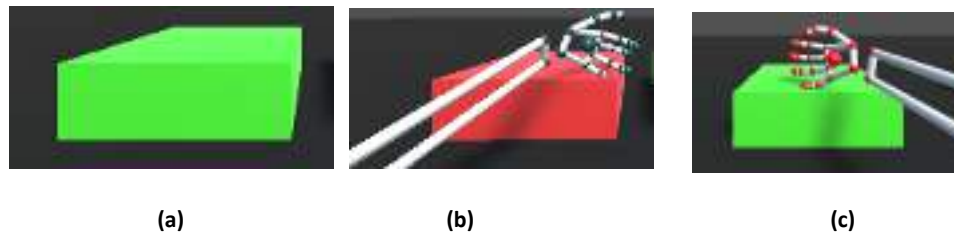


Figure 19: Select a game object by touching it (the game object's color becomes red when it is selected), (a) the gameobject; (b) use left hand to select, (c) use right hand to deselect.

Another possibility is to use the ray defined by the direction of one finger for the selection, while the ray defined by the corresponding finger in the other hand can be used for the deselection (see figure 20 below).

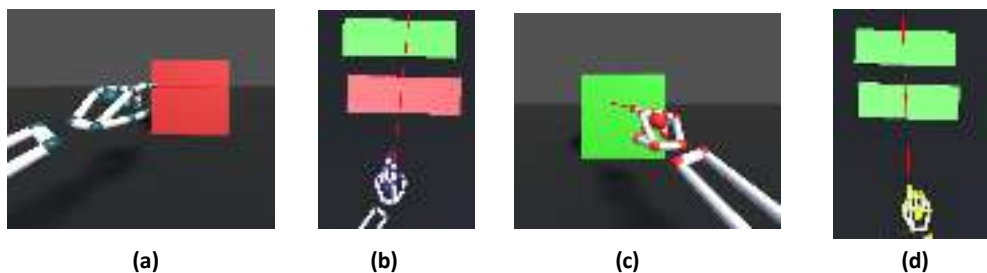


Figure 20. Select and deselect a game object by casting a ray. The game object's color becomes red when it is selected; (a) and (c) are front views, (b) and (f) are top views. (a) and (b) the game object in sight is selected by left hand while the hidden game object is not selected; (c) and (d) the game object in sight is deselected.

These two metaphors have some disadvantages: sometimes it is not easy to touch a control point, while the leap motion controller can not track the finger when it is occluded e.g., by the rest of the hand. Thus the above methods are not effective nor precise for the selection.

To avoid these problems, we adopted a combination of gaze and voice command for the selection. Gaze selection procedure considers a ray from the camera based on the camera's view direction. If the user moves the head and the ray hits a control point, the control point is highlighted assuming a yellow outline. Then, if the user says "select" while the control point is highlighted, the control point will be included in the *selected control points* list and becomes red. The procedure for the selection/deselection of a control point is (see figure 21):

1. The user looks at a control point that he/she wants to select; this control point becomes selectable and is highlighted;
 2. The user says “select” to select this control point. Once the control point has been selected, its color will change to red;
 3. The user looks at a control point that he/she wants to deselect (red sphere) and it is highlighted; this control point becomes deselectable with the voice command “discard”
- Applying the above commands, the user may select or deselect as many control points as he/she wants.

Figure 21 shows the selection flowchart of a set of control points.

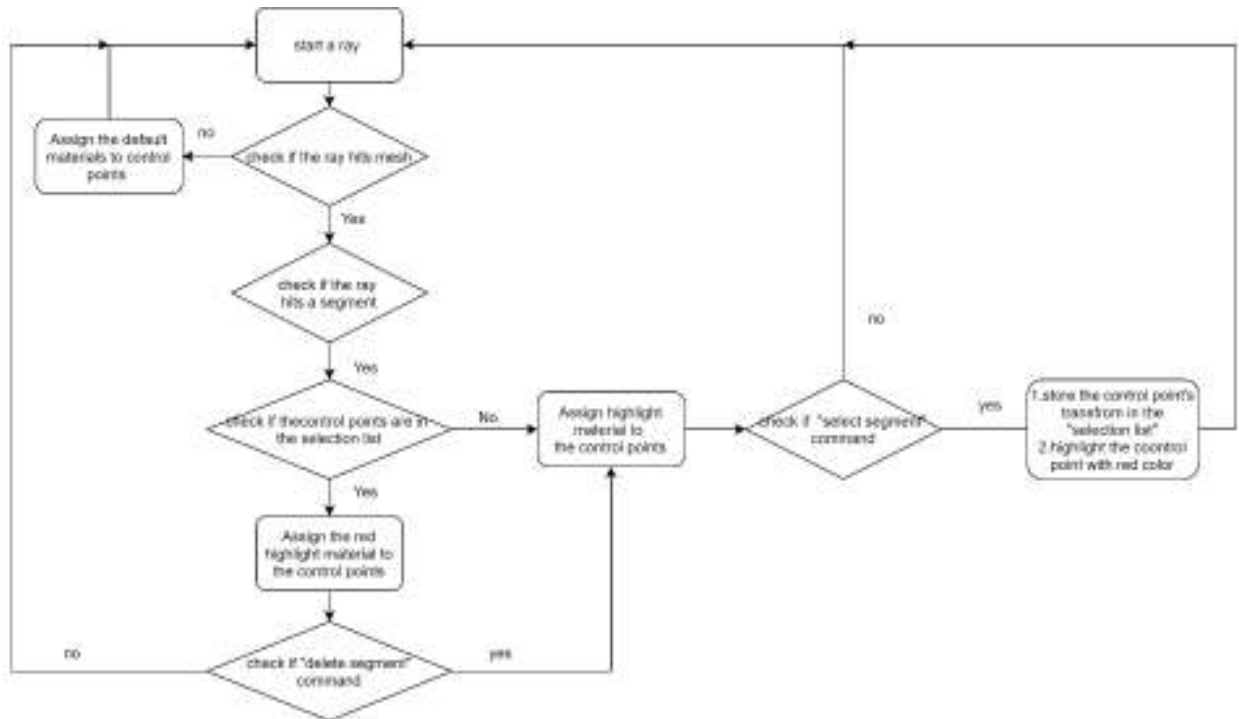


Figure 21: The flow chart of the selection or deselection of a control point

In addition to the single control point selection repeated as many times the user wants, there is also the possibility of selecting simultaneously all the control points corresponding to a specific annotated segment. To do this, the user has to follow the steps below:

1. Select the level of the annotation hierarchy to which the segments the user is interested in belong to.
2. Gaze at the target mesh segment. The related visible control points are circled in yellow while the not visible (hidden by the shape itself) are colored in yellow (see fig.22.b) .

3. Use the voice command “select segment” once the corresponding control points are highlighted in red to confirm the selection (see fig. 23).



(a)



(b)

Figure 22: The selection of a set of control points associated to the “thumb” segment (a) : control points are hit with the gaze and highlighted in yellow (b)



(a)



(b)

Figure 23: (a) a set of control points are highlighted when the ray from camera hits the model (b) a set of control points are selected when the voice command “select segment” is made

Using the voice command “discard segment” when the ray is hitting a segment that has some selected control points associated, the control selected points will be de-selected and the default highlight materials will be assigned to them (i.e. the material of the segment’s color but with yellow

highlight on the surface of control points. Transversal selection or deselection of control points at different levels is possible, for example, the user can select the pinky at level 1 and deselect the control points on the pinky's nail at level 2.

3.2.3. The translation of the control points

The production of some types of deformations, like for instance the elongation of one part, can be obtained by translating the position of some control points. This can be obtained in different ways according to the interaction mean.

3.2.3.1. Mouse version

In our application, the selected control points are translated by moving the mouse and following its position.

In some cases, the deformation of an object has to undergo some constraints inherited by other objects, or by dimensional conditions. To test and simulate this situation, we included the metaphor of the “room”, i.e. the deformed object has to stay within the room walls. In this way, we have to limit the translation of the 3D models representing handles inside the room. In figure 24, we can see a *selected control points* game object being dragged from the position of figure 24a to the position of figure 24e whose displacement is stopped when the game object hits an obstacle at the position of figure 24e.

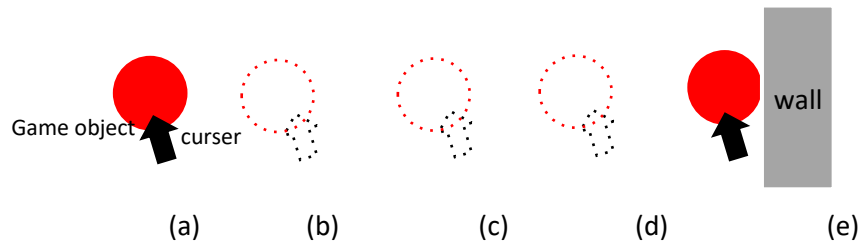


Figure 24: Schematic diagram of dragging a game object with cursor: (a) is the start position; (b), (c), (d) are the intermediate positions of the cursor and the game object; (e) is the position when the game object hits a wall

The steps of the algorithm for the constrained dragging of a game object (see figure 25):

- 1) Initialization: when the mouse clicks on a game object, record the cursor's position as “previous position” and assume “free movement” state;
- 2) While the mouse drags the game object, the “free movement” state (i.e., it does not collide with obstacles) is constantly updated. As long as it is true, the game object follows the cursor updating its position: When its state is false, the following operations done to compute the possible directions of movements of the object:

- i) $\text{allowed direction} = \text{previous position} - \text{current cursor's position}$ and $\text{not allowed direction}$ as the normal to the hidden object;
- ii) update the previous position with the current cursor's position value;
- iii) compute an angle α between the “allowed direction” and the “not allowed direction” (see figure 26); if the angle is in a certain range , allow the user to drag the game object.

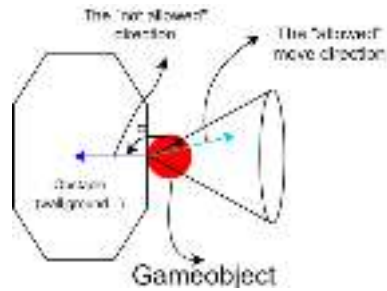


Figure 25: The possible dragging direction of the game object while collision happens

- 3) Set the state variable “free movement” to true (false), when the game object enters (leaves) the collision between game object and barrier.

The “free movement” state updating exploits an Unity asset, i.e. a mesh collider, that is able to detect objects collision. To guarantee the above behavior when a mesh is deformed, models have to be equipped with own mesh colliders and the one of the deformed model has to be updated accordingly to the mesh (see figure 27.c).

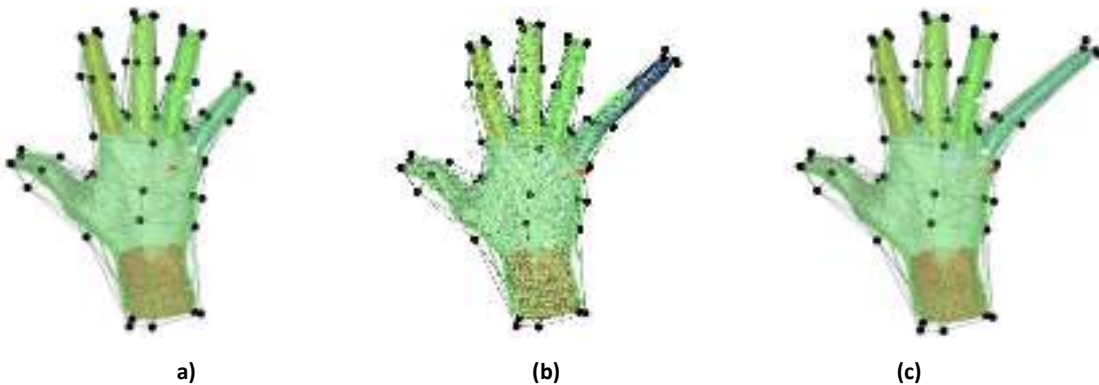


Figure 27: (a) the default green mesh collider (the model is not deformed); (b) the mesh collider is not deformed with the model (the model depicted in blue is deformed); (c) the mesh collider is deformed with the model at real time (the model is deformed)

3.2.3.2. Leap Motion version

With Leap Motion, the selected control points are moved according to the forefinger of the right hand. The steps for achieving the movement are:

1. get the position of the index fingertip: this will be the starting position;
2. track the movement of the fingertip: a vector is extracted based on the actual position and the starting one;
3. move the selected control points in the direction and with the magnitude defined by the just extracted vector.

3.2.4 Rotation

In this project, rotation deformations have been implemented only on the mouse version. .

3.2.4.1. Mouse version

The steps for rotating a game object:

1. The user clicks the left mouse button and drags the mouse: the clicking position is recorded as s and the release one is recorded as r ;
2. Get the origin (o_c) and direction (n_c) of the camera and compute $n_p = r - s$;
3. Extract the rotation axis $a = \frac{n_c \times n_p}{|n_c| \cdot |n_p|}$ (see the figure 28)
4. Get the angle as $\alpha = \cos^{-1} \left(\frac{(s - o_c) \cdot (r - o_c)}{|s - o_c| \cdot |r - o_c|} \right)$.
5. Take the center of the game object c_{GO}
6. Rotate the game object using the computed, centered in the game object's center axis and angle.

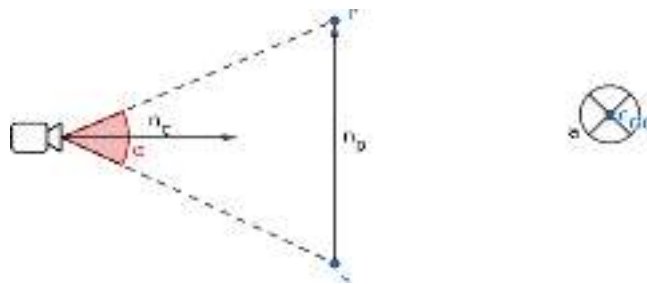


Figure 28: The scheme of the rotation

Figure 29 shows the flow chart for the rotation.

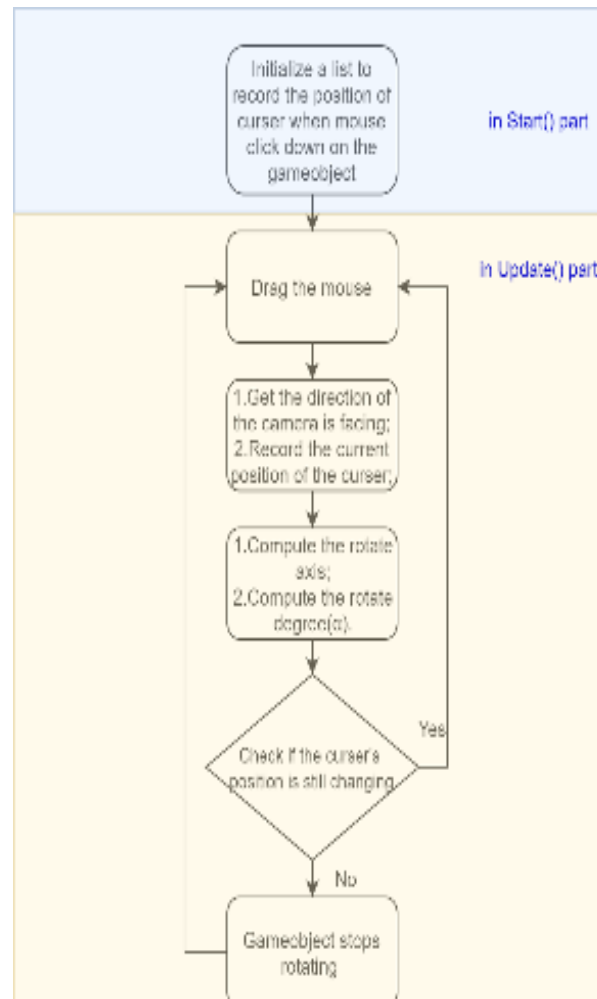


Figure 29: The flow chart of rotating the game object with a mouse

3.2.5. Scale the model

The scale function is applied to the whole object.

3.2.5.1. Mouse version

Since we want the model and cage to be fully synchronised, scaling the model means scaling the cage. Then, we can obtain the correct modification of the model computing the usual matrix multiplication $M' = B \cdot G'$. Firstly, we should find a point as the scaling center SC . In figure 30, the scaling of a set of vertices is shown: if we take as example the point A, we can obtain the vector

$\mathbf{a} = \mathbf{A} - \mathbf{SC}$. We expect the magnitude of this vector to enlarge (or scale), by a constant coefficient (2 in the image). The new point position \mathbf{A}' is defined as $\mathbf{A}' = \mathbf{SC} + 2 \cdot \mathbf{a}$. In this project, the scaling center is given by the centroid of the points to be scaled, i.e., $\mathbf{SC} = \frac{1}{n} \cdot \sum_{i=1}^n (\mathbf{p}_i)$.

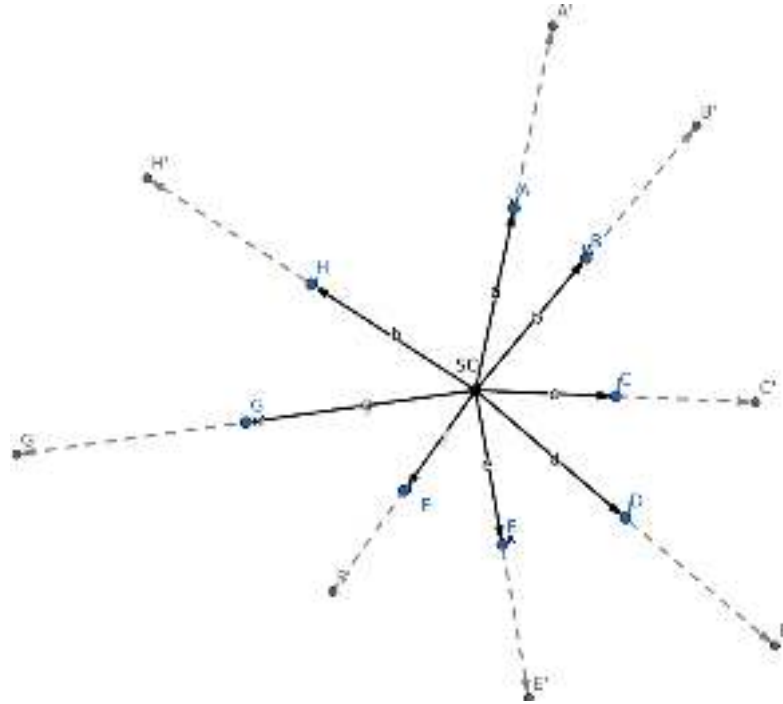


Figure 30: A vertex on the model before and after scaling

A slider has been added in the 3D scene to input the scale ratio by simply dragging the handle of the slider.

3.2.5.2. Leap motion version

After selecting a level, the user can scale the model by:

- Say “scale” to activate the mode;
- Clench a leap hand into a fist to reduce the model size or fully open a leap hand to enlarge the model (still, the model is constrained to stay inside the room, so that if the model’s mesh hits a wall or the ground, the user cannot enlarge the model any further).
- Move away the leap hands from the leap controller and say “stop scaling” to deactivate scale mode.

4. Conclusion and future work

The work here described represents a first effort in bringing modelling functionalities within the immersive environments. Here only the cage-based mesh deformation has been addressed defining interaction functionalities, both with mouse device and leap motion integrated with voice commands and gaze tracking. The purpose was to provide functionalities to easily interact with the model and deepen understanding of the model features; understand some basic principles of cage-based mesh deformation and how to apply it; enable users to select a level and do a series of modifications to the model.

Future work includes the use of Head Mounted Display for the immersive virtual reality integration and the extension of the modification functionalities. Another branch of work will relate to the inclusion of other types of barycentric coordinates (such as PMVC, HC, etc.) to carry out different cage-deformations and compare them. Save functionalities will be added to allow several modelling sessions; also capabilities for adding annotations and comments to the modifications will be considered. Finally, we intend to carry out user experimentations to evaluate the usability of the system and possibly compare it with other projects of the same type.

References

- [Ake] Akers, J.. *TryDesVR: Virtual Reality Based Interactive Mesh Deformation*.
- [CGM19] Cordeiro E., Giannini F., Monti M.: A survey of immersive systems for shape manipulation. *Computer-Aided Design and Applications* 16, 6 (2019), 1146 – 1157. URL: http://cad-journal.net/files/vol_16/Vol16No6.html, doi:<https://doi.org/10.14733/cadaps.2019.1146-1157>. 2
- [FV08] D. Faas, J. Vance. *Interactive Deformation Through Mesh-Free Stress Analysis in Virtual Reality*. 2008.
- [Gra] Gravity Sketch. URL: <https://www.gravitysketch.com/>
- [GWF18] Z. Gao, J. Li, H. Wang, G. Feng. 2 DigiClay: An Interactive Installation for Virtual Pottery using Motion Sensing Technology. In *Proceedings of the 4th International Conference on Virtual Reality (ICVR 2018)*. Association for Computing Machinery, New York, NY, USA, 126–132. DOI:<https://doi.org/10.1145/3198910.3234659>
- [HW18] He, C., and C. Wang. "A Survey on Segmentation of 3D Models." *Wireless Personal Communications*, vol. 102, no. 4, 5 Feb. 2018, pp. 3835–3842, 10.1007/s11277-018-5414-1
- [Lea] Leap Sculpturing. URL: <https://gallery.leapmotion.com/sculpting/>
- [Mak] MakeVR Pro, 2017. URL: <https://www.viveport.com/9e94a10f-51d9-4b6f-92e4-6e4fe9383fe9>
- [Min] MindeskVR, 2015. URL: <https://mindeskvr.com/>
- [MCGF19] Mendes, D., Caputo, F.M., Giachetti, A., Ferreira, A. and Jorge, J. (2019), A Survey on 3D Virtual Object Manipulation: From the Desktop to Immersive Virtual Environments. *Computer Graphics Forum*, 38: 21–45. doi:[10.1111/cgf.13390](https://doi.org/10.1111/cgf.13390)
- [NS12] J. R .Nieto, A. Susín. "Cage Based Deformations: A Survey." *Deformation Models*, 30 Oct. 2012, pp. 75–99, 10.1007/978-94-007-5446-1_3. Accessed 31 Aug. 2020.
- [Ocu] Oculus Medium. URL: <https://www.oculus.com/medium/>
- [PCDS19] Ponchio, F.,M. Callieri, M. Dellepiane, R. Scopigno "Effective Annotations Over 3D Models." *Computer Graphics Forum*, vol. 39, no. 1, 15 May 2019, pp. 89–105, 10.1111/cgf.13664.
- [RASFO7] F. Robbiano, M. Attene, M. Spagnuolo and B. Falcidieno, "Part-Based Annotation of Virtual 3D Shapes," 2007 International Conference on Cyberworlds (CW'07), Hannover, 2007, pp. 427-436, doi: 10.1109/CW.2007.8.
- [SMS20] A. Scalas, M. Mortara, M. Spagnuolo, A pipeline for the preparation of artefacts that provides annotations persistence, *Journal of Cultural Heritage*, Volume 41, 2020, Pages 113-124, ISSN 1296-2074
- [Soli] <https://atap.google.com/soli/> last accessed 6.11.2020
- [Track] <https://www.inition.co.uk/product/ascension-3d-guidance-trakstar/> last accessed 6.11.2020
- [VMW15] A. Vaxman, C. Müller, O. Weber. "Conformal Mesh Deformations with Möbius Transformations." *ACM Transactions on Graphics*, vol. 34, no. 4, 27 July 2015, pp. 1–11, 10.1145/2766915
- [VR15] Vinayak, Ramani K.: A gesture-free geometric approach for mid-air expression of design intent in 3D virtual pottery. *Computer-Aided Design* 69 (2015), 11 – 24. URL: <http://www.sciencedirect.com/science/article/pii/S001044851500086X>, doi:<https://doi.org/10.1016/j.cad.2015.06.006>.

[ZZWCY19] H. Zhu, X. Zuo, S. Wang, X. Cao, R. Yang “Detailed Human Shape Estimation from a Single Image by Hierarchical Mesh Deformation.” *Arxiv.Org*, 24 Apr. 2019, arxiv.org/abs/1904.10506. Accessed 31 Aug. 2020.

Recent titles from the IMATI-REPORT Series:**2019**

19-01: *Feature curves, Hough transform, characteristic elements*, C. Romanengo, S. Biasotti, B. Falcidieno.

19-02: *On expansions and nodes for sparse grid collocation of lognormal elliptic PDEs*, O.G. Ernst, B. Sprungk, L. Tamellini.

19-03: *Augmented Reality in manufacturing engineering*, B. Bonino, F. Giannini, M. Monti.

19-04: *Parametric shape optimization for combined additive-subtractive manufacturing*, C. Altenhofen, M. Attene, O. Barrowclough, M. Chiumenti, M. Livesu, F. Marini, M. Martinelli, V. Skytt, L. Tamellini.

19-05: *Architecture of a client-server platform for IMATI cultural heritage applications*, A. Repetto, C.E. Catalano, M. Spagnuolo

2020

20-01: *Compressive isogeometric analysis*, A. Brugiapaglia, L. Tamellini, M. Tani.

20-02: *A gradient-based optimization method with functional principal component analysis for efficient structural topology optimization*, A. Montanino, G. Alaimo, E. Lanzarone

20-03: *Nota sull'evoluzione temporale dell'epidemia causata dal SARS-CoV-2 in Italia. Una analisi dei dati nazionali diffusi dalla Protezione Civile*, G. Buffoni.

20-04: *Cage-based Mesh deformation in VR*, Y. Zhu, K. Lupinetti, A. Scalas, F. Giannini, M. Monti, M. Mortara, R. Lou, M. Spagnuolo