# An Agent-Based Paradigm for Virtual Modeling

Julián Conesa[1], Jorge D. Camba[2*], José Ángel Aranda[3], Manuel Contero[4]

[1]Departamento de Expresión Gráfica, Universidad Politécnica de Cartagena
Campus Muralla del Mar, C/ Doctor Fleming s/n, 30202 Cartagena, Spain
julian.conesa@upct.es

[2]Department of Computer Graphics Technology, Purdue University
401 Grant St. West Lafayette, IN 47907, USA
jdorribo@purdue.edu
*Corresponding Author

[3]Departmento de Ingeniería Gráfica
Universitat Politècnica de València
Cno. de Vera s/n, 46022 Valencia, Spain
jaranda@upv.es

[4]I3B - Universitat Politècnica de València
Cno. de Vera, s/n. 46022 Valencia, Spain
mcontero@upv.es

# An Agent-Based Paradigm for Virtual Modeling

**Abstract**

Recent advances in graphics hardware technology are enabling the development of increasingly more sophisticated virtual reality (VR) applications. However, the representation of 3D virtual models in this medium requires finding a proper balance between visual quality and computational complexity. Virtual reality-based software should be optimized in a manner that the functions responsible for representing and rendering the 3D scene do not execute any tasks that may cause delays in the rendering process. In this paper, we describe, implement, and validate a multi-agent architecture for VR-based geometric modeling that enables managing user interaction and the geometric database independently from the representation of the virtual model. Our results show that the proposed approach can significantly improve rendering refresh rates when compared to conventional methods.

**Keywords**: agent-based system, virtual modeling, VR rendering.

## 1 Introduction

Recent advances in emerging visualization technologies such as Virtual Reality (VR) are changing the way 3D content is viewed and experienced. VR is a visualization technology in which the view of the physical world is fully replaced by an artificial environment (Jerald, 2015). It can be experienced through different modalities, such as Cave Automatic Virtual Environment (CAVE) systems (Cruz-Neira et al., 1993), power walls, and individual VR headsets. VR experiences generally involve an immersive stereoscopic real-time view of the environment, which is enabled by a head tracking system that responds to the user's changes in position and orientation to recalculate the view and render the virtual content appropriately. In some experiences, there is also auditory, haptic, and tactile feedback when the user interacts with the elements of the virtual world.

VR technologies have been used across a wide range of disciplines. In design and engineering, VR has been applied to the visualization of design concepts and processes (Schmitt, 1993; Shibano et al., 2001; Messner et al., 2002; Whisker, 2003), analysis of parts, equipment, and processes (Opdenbosch and Hastak, 1994; Lipman and Reed, 2000; Ryken and Vance, 2000), design reviews (Bassanino et al., 2010), interior architecture processes (Bannova et al, 2019) and design education (Camba et al., 2017; Nisha, 2019), to name a few. In these scenarios, VR provides a mechanism to view virtual models in context and experience the design in a more natural and intuitive manner.

Despite the remarkable advances of VR technology in recent years, there are still challenges that must be overcome before the technology is widely adopted in industrial settings. For example, ergonomically, VR headsets tend to be heavy and bulky, and in some cases tethered, which limits the user's movements and affects comfort. Haptic and tactile feedback mechanisms, which are critical for applications with high levels of presence, are also limited.

From a content standpoint, the lack of intuitive authoring tools, user interfaces, and interaction mechanisms for creating, manipulating, and editing VR content also hinders adoption. Importing native Computer-Aided Design (CAD) data into a VR system requires file conversions and optimizations, a process that is often time-consuming and error prone (Zhong et al., 2005). Additionally, any changes to the native CAD file require reconverting the geometry so it can be consumed by the VR system. Furthermore, the conversion process is unidirectional. The fact that any changes made to a model in the VR environment cannot be transferred back to the CAD file may result in data redundancy and inconsistencies (Zorriassatine, 2003). In this regard, some authors have explored 3D modeling as a VR-based process where geometry is created directly in the virtual environments (Chu et al., 1997; Arangarasan & Gadh, 2000; Cappello et al., 2007; Ladwig et al., 2017). Several commercial tools such as Masterpiece Studio, Gravity Sketch, or Google Tilt Brush, have also been developed, mainly for artistic purposes.

The increasing geometric complexity, precision, and level of detail of the 3D models that are used in VR applications require powerful hardware and sophisticated techniques to be able to render the scene in real time, which is particularly challenging in multiuser network scenarios, where large amounts of data need to be transferred over the network. Failure to provide an appropriate rendering performance can have significant effects in the quality of the VR experience. In extreme cases, it can induce "simulator sickness," a syndrome that may cause discomfort, dizziness, disorientation, fatigue, and nausea (Kolasinski, 1995).

The visual aspects of VR have a high computational cost in both the CPU and the GPU, which can cause problems during the rendering process. Limited fill rate is a common problem in the GPU whereas in the CPU problems typically stem from the number of batches that need to be rendered. Occasionally, these problems may be caused by certain scripts which can delay the rendering of the scene. In this paper, we describe a novel multi-agent approach for 3D modeling applications in VR environments in which dedicated agents free up the class responsible for rendering from executing other tasks that may require significant computational time. The paper is structured as follows: first, we review the relevant work in the areas of 3D modeling in VR environments and multi-agent systems. Next, we describe the proposed virtual modeling paradigm and its multi-agent architecture. We examine the structure, the different modules, and the relationships between these modules in detail. Finally, we present the results of a study to determine the gains provided by our approach in terms of rendering times.

# 2 Related Work

From a usability standpoint, the use of 2D tools for creating 3D content is not the most intuitive or natural approach to modeling. The complexity of curves, surfaces, and other geometric elements that are commonly used in 3D design quickly expose the challenges and limitations of working in the intrinsically two-dimensional environments enabled by traditional computers and devices. Alternatively, VR environments provide a more efficient and friendly setting to build 3D models by allowing users to manipulate geometry and move around the virtual world as they would normally do in the real one. This paradigm is more natural and easier to learn and use than traditional 3D modeling tools, particularly for inexperienced users (Rosales et al., 2019).

The notion of drawing sketches and building 3D objects in mid-air may sound like a natural approach to geometric modeling, but it is actually quite challenging, as reported by Arora et al. (2017). In their study, the authors determined that the lack of a physical medium during sketching may cause inaccuracies when modeling in VR. In this regard, visual guides may

improve positional accuracy, but they can also affect the aesthetic quality of the end result Arora et al. (2017).

In the context of Computer-Aided Design and Engineering, where geometric accuracy and precision are critical, VR-based technologies offer significant advantages over traditional CAD tools (Trika et al., 1997). First, the design process is simplified as geometry is more accessible when viewed within an immersive environment. This is particularly important when manipulating small details that may not be easily reachable such as holes, pockets, and objects with many internal elements. Second, VR-based modeling provides an intrinsic understanding of the part's manufacturability during the modeling process, as designers create and manipulate geometry directly on the surfaces of the part, a task that relies on the designer being able to reach those surfaces, similar to how a machine would manufacture the part. Finally, VR-based modeling approaches can also provide suggestions and feedback regarding manufacturing process planning tasks (Trika et al., 1997).

VR-based modeling can also help eliminate problems and errors stemming from file conversions between native CAD formats and formats that are used in VR applications. To this end, researchers Zhong et al. (2005) proposed a series of recommendations to improve product modeling processes. The authors suggest that the entire modeling process as well as any subsequent modifications to the geometry should take place directly within the VR environment. However, some of the limitations of current VR-based modeling tools may affect creativity and design exploration. For example, authors Keefe and Jackson (2016) pointed out the challenge to accurately control geometry during freehand sketching and modeling tasks, and proposed a solution based on starting 2D sketches outside the VR environment which are then used as input for building 3D content. The lack of accuracy seems to be a common problem in VR-based modeling environments (McGraw et al., 2017; Machuca, et al., 2018). In terms of user experience, researchers Cordeiro et al. (2019) proposed a series of guidelines for developing more natural and interactive modeling tools in immersive environments. Likewise, several studies have been conducted to determine the most effective and intuitive navigation, menus, and interaction methods in VR (Monteiro et al., 2019; Soler-Dominguez et al., 2019). In this paper, we propose a VR-based 3D modeling system for creating accurate geometry directly within the virtual environment. Geometry is created and manipulated through a set of tools which are available to the user via a virtual menu and can be accesses via the VR controllers.

The second main contribution of our paper is a multi-agent approach to the architecture to optimize VR rendering times. Although there is not a universally accepted definition of what constitutes an agent, the term can be understood as a lightweight and independent software entity that performs a simple operation on a particular element (in our case, a VR scene model). Agents may interact and work with other agents, defining a Multi-Agent System (MAS), which can be used to solve complex tasks that are difficult or impossible for an individual agent or a conventional system. In this regard, the whole is greater than the sum of its parts.

Autonomy is the central notion of agency (Oliveira et al., 1999). For the purposes of our study, the following characteristics of agents in a multi-agent system are relevant (Wooldridge, 2002, Tessier et al., 2002):

- Agents are autonomous and partially independent from each other
- Agents can interact and communicate with other agents

- No agent has a full global view of the system
- No agent is designated as controlling

Multi-agent systems have been the subject of study for several decades and have been applied to a variety of problems. In the field of computer graphics, they have been used in AI-based game elements such as navigation and motion planning for non-player characters (Bleiweiss, 2009), as well as the design of particle systems (Xiao et al., 2011), crowd simulations and shape constrained animations (Chen et al., 2019).

In the area of rendering, multi-agent systems have been used to simulate non-photorealistic styles through stroke-based rendering techniques such as stippling and hatching (Schlechtweg et al., 2005). Researchers have also leveraged the intrinsically distributed nature of multi-agent systems and their intelligent interaction properties to optimize rendering processes. For example, Rangel-Kuoppa et al., (2003) developed a JADE-based multi-agent platform to distribute rendering tasks across a network. However, researchers Gonzalez-Morcillo et al., (2007) argued no "agent-typical" mechanism is used in this implementation, and developed their own system, which applies multi-agent principles to 3D realistic rendering optimization (Gonzalez-Morcillo et al., 2007).

Research with multi-agent systems in virtual environments has mainly focused on real time interaction and simulations systems for training and safety applications (Wang et al., 2013; Sharma et al., 2014) as well as interactive prototyping (Chevaillier et al., 2000). However, the application of multi-agent systems to optimize real time rendering processes in immersive environments remains unexplored.

In this paper, we describe and demonstrate a novel multi-agent architecture for VR-based 3D modeling that significantly improves the graphical representation of 3D models and reduces the computational load of both the CPU and GPU.

# 3 A virtual reality-based modeling paradigm

As part of this work, we developed a geometric modeling application for a virtual reality platform (see Fig 1). The application allows users to interact with virtual content via the trigger buttons of the VR controllers and a menu located over the left controller and visible in the virtual environment. The commands can be run by selecting them with the right controller.

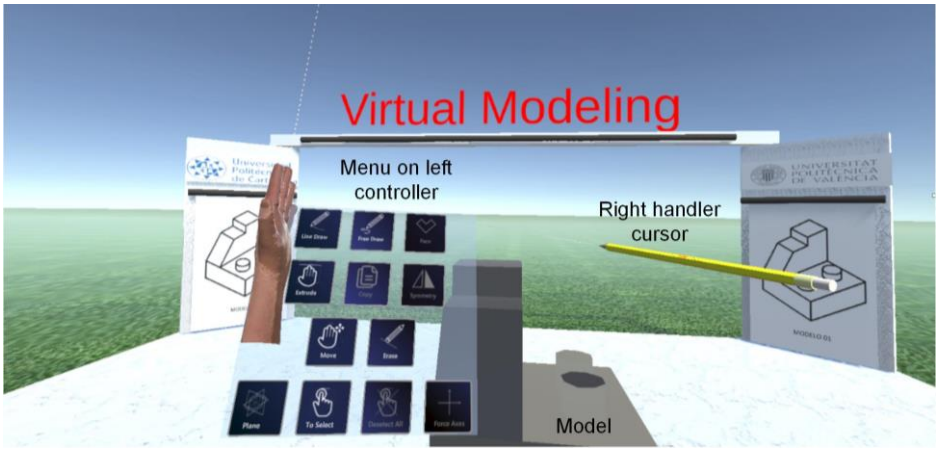

Fig. 1. Application environment.

Our system allows the creation of a wide variety of geometric models. The available commands are shown in Fig. 2 grouped by color. Green represents basic auxiliary commands such as “select entity” and “define work plane;” blue represents commands for creating new geometry; and orange represents commands for editing existing geometry. The modeling tools produce a boundary representation (B-Rep) comprised of vertices, edges (defined by two vertices), curves (defined by a set of edges) and surfaces (defined by a set of edges and curves).

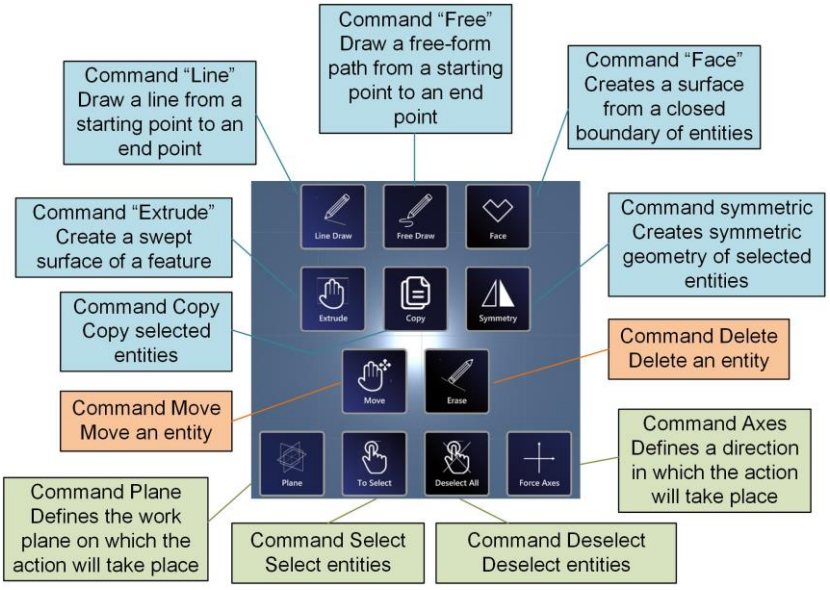


Fig. 2. Modeling tools and commands.

The mode of operation of the commands is based on the type of virtual contact initiated by the user (with the controllers) with the geometric entities that have been modeled in the VR environment. To provide visual feedback to the user every time a contact is identified, the entity is highlighted in red. When contact occurs at one end of a geometric entity, a red sphere and a textual note will be displayed on the corresponding vertex. If the contact occurs on an edge, curve, or surface, the entire geometric entity will turn red. Additionally, some commands require the selection of some entities, which will be shown in blue to distinguish them from unselected entities. For example, in Fig. 3, middle unselected edges are shown in black (the default color), the edge at the bottom is shown in blue after it was selected, and one vertical edge is shown in red after contact with the right controller was identified. In the following sections, we describe each command of the application in detail.

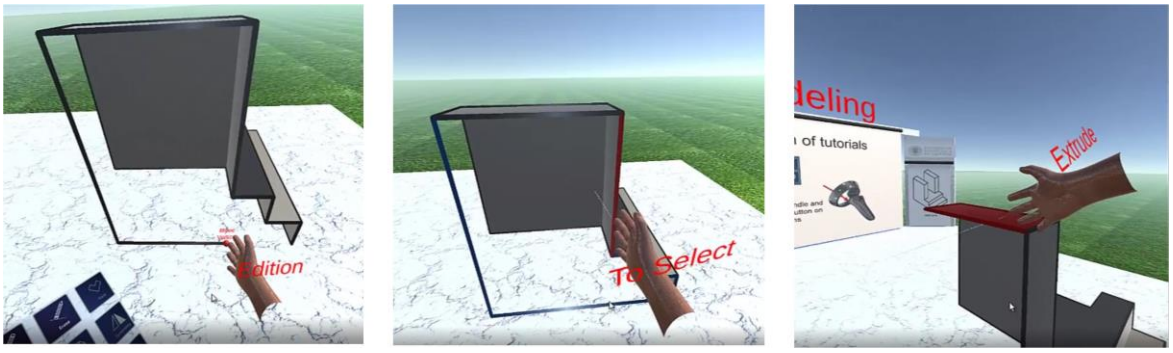


Fig. 3. Contact of the controller with a vertex (left), an edge (middle), and a surface (right) during the execution of a command.

## *3.1 Auxiliary Commands*

In this section, we describe the commands that were implemented to support the 3D modeling process. First, the "plane" command allows users to define a working plane, which can be set to horizontal, frontal, and profile with respect to the virtual 3D space, as well as any custom plane defined by three noncollinear points. A delete action is also available on the button to delete any existing work planes.

While defining a horizontal, frontal, or profile orientation, the plane remains linked to the left controller until the trigger button is pressed. At that point, the coordinates of the cursor are used to fix the position of the work plane. To complete the geometric definition of the plane, two additional points are used to define the unit vector that establishes the horizontal and vertical directions on the plane. Similarly, to define a custom plane using three noncolinear points, the user must fix the first point so the system will preview a plane that passes through that point and the orientation varies based on the position of the left controller. When the second point is set, the orientation of the plane will be constrained so that it passes through the line that connects the two points (the orientation still depends on the position of the left controller). Finally, the plane is fully defined after the third point is set.

When a plane is defined, the cursor that represents the right controller will be projected onto the plane. Therefore, at that point, all commands and modeling operations will be applied on the plane. The work plane can also be used as reference geometry for certain modeling operations such as symmetry. Four work planes defined during the modeling process of a 3D part are illustrated in Fig. 4. Note how the cursor that corresponds to the right controller is linked to the work planes.

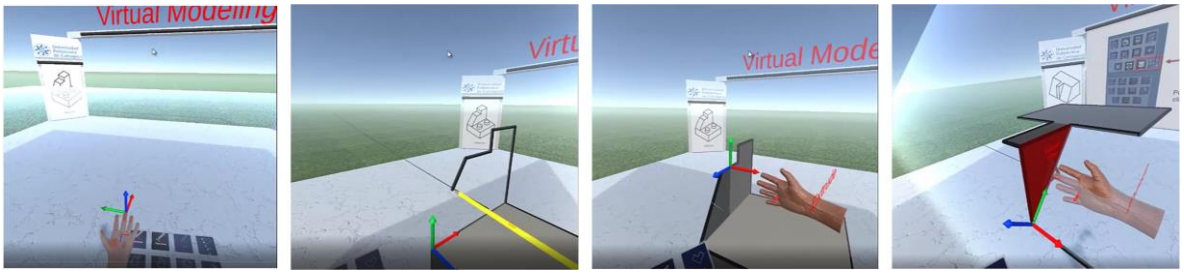

Fig. 4. Work planes. From left to right: horizontal, profile, frontal, and custom plane.

Once a work plane has been defined, the "axes" command can be used to set direction of the cursor linked to the right controller. The X-direction is defined by the first two points used to define the work plane, and the Y-direction is perpendicular to it. The icon that represents the axes is shown in Fig. 4. The X-axis is shown in red and the Y-axis in blue.

Finally, the "select" command is used to select different geometric entities in the 3D model. The selection action is necessary to run the "copy" and "symmetry" commands. To select a specific geometric element, the user needs to press the trigger button in the right controller while the cursor is in contact with that element. If a previously selected object is selected again, a deselection occurs. In addition, running the "deselect" command will deselect all the currently selected entities.

## *3.2 Commands for creating geometry*

In this section, we describe the commands that are used to generate 3D geometry:

- Line: the line command is used to draw straight lines by pressing and holding the trigger button of the right controller. The first endpoint is set at the location of the cursor and the second endpoint can be set by moving the controller to the desired position and releasing the trigger button (see Fig 5, left).
- Free: this command allows the creation of a complex shape as a series of connected lines. The command starts by pressing the trigger button of the right controller, which sets the first endpoint of the first line to the current position of the cursor. The second endpoint is set by moving the controller to the desired position until the distance between the two endpoints is greater than a predefined threshold (determined experimentally), which sets the coordinates of the endpoint and starts a new line at that point. The process is repeated for every line in the sequence (see Fig 5, right).

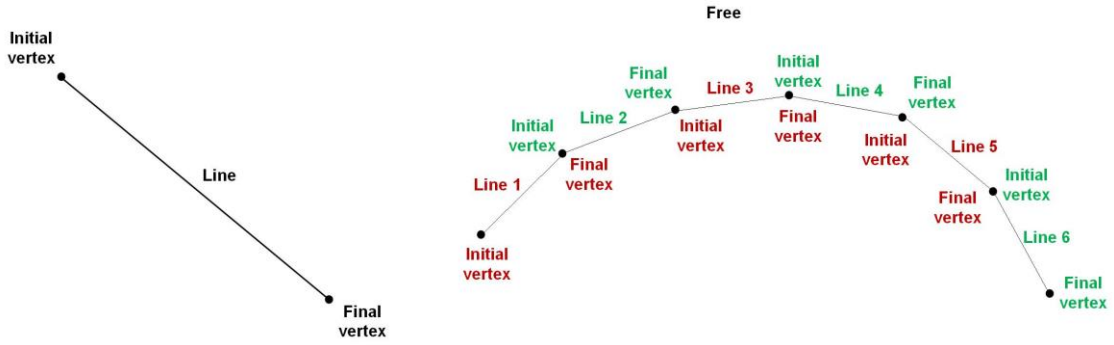


Fig. 5. Geometric entities created by the line command (left) and free (right).

- Copy: the copy command requires some entities in the model to be selected. By pressing and holding the trigger button of the right controller, a copy of the selected objects is created, which follows the position of the controller until the trigger button is released. (see Fig 6, left).
- Symmetry: to run this command, the user must have previously defined a work plane and selected at least one geometric entity. The command is executed by pressing the trigger button of the right controller over the corresponding menu button. The result is a new set

of geometric entities which are symmetrical to the original geometry with respect to the work plane (see Fig 6, right).

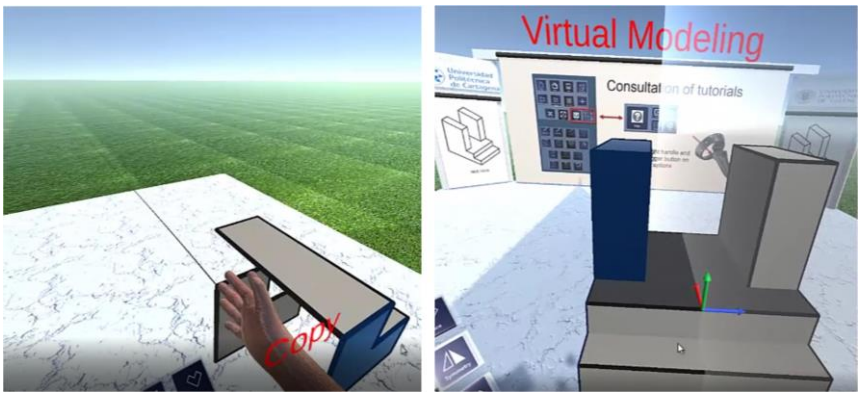


Fig. 6. Copy (left) and Symmetry (right) commands.

- Face: to run this command, the user must have previously selected at least one geometric entity. The command is executed by pressing the trigger button of the right controller over the corresponding menu button, which runs an ear clipping algorithm to determine faces from the selected geometric entities (Eberly, 2008).
- Extrude: the extrude command creates swept surfaces from vertices, edges, curves, or surfaces. The command is executed by placing the cursor that corresponds to the right controller over the desired geometric entity and pressing and holding the trigger button. The extruded geometry can be defined by moving the controller while holding the trigger button. If the extrusion is applied to a vertex or a line, a new surface is created. If applied to an existing surface, then a copy of the existing surface is created along with a series of surfaces equal to the number of entities that comprise its contour, thus creating an enclosed volume (see Fig. 7).

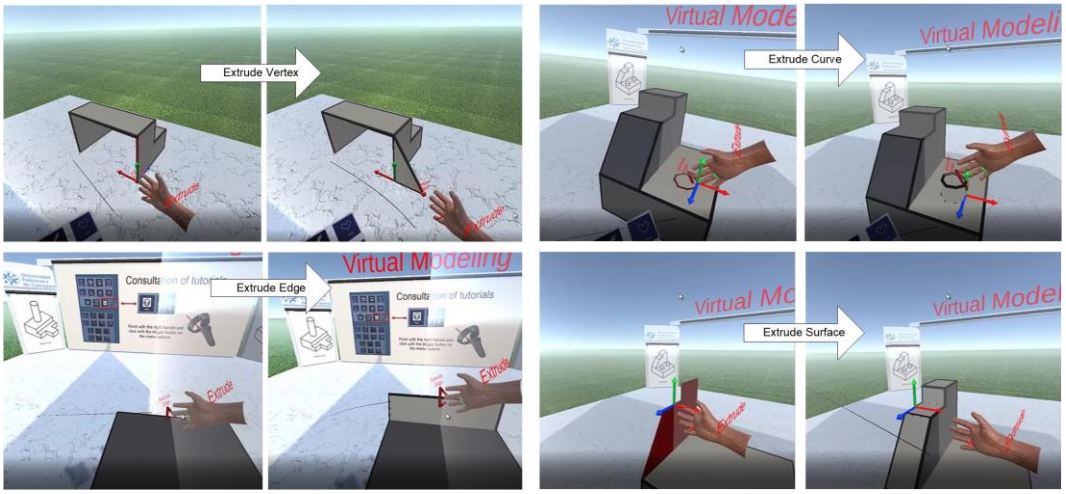


Fig.7. Surfaces created by the “extrude” command.

### *3.3 Commands to edit existing geometry*

Two commands are used to edit existing geometry: delete and move. The delete command is used to delete existing geometry by pressing the trigger button of the right controller over the desired entity. The move command is used to change the position of an existing element by pressing and holding the trigger button of the right controller, moving the controller, and finally releasing the button when the object is at the desired new location.

## 4 Agent-based architecture

The paradigm described in the previous section was implemented and assessed on a multi-agent architecture based on the Unity game engine. Unity provides a GameObject class which, in its mode Empty, is the most generic class available. We can use the Empty GameObject class to connect scripts that derive directly from Unity's base class "MonoBehaviour," thus associating behaviors and a messaging system to each Empty GameObject. As a result, each Empty GameObject can be considered an independent agent where the behaviors "one shot" and "cyclic" are supported by the functions "start" and "update," which derive directly from the MonoBehaviour class. The "cyclic" process is accomplished via a call for each frame in the virtual scene.

The architecture proposed in this paper is structured in three main modules:

1) The Interface Agent Modules (IAM) is comprised of a single agent, the Interface Agent (IA), which is responsible for managing the user interaction, i.e. the actions with the controllers and the menus.
2) The Broker Agent Modules (BAM), which is comprised of a single agent, the broker agent (BA), and is responsible for connecting IAM with the Entities Agents Module (EAM).
3) The Entities Agents Module (EAM) is comprised of a Manager Agent (MA) in charge of coordinating all the other agents that are part of EAM: Vertex Agents (VA), Line Agents (LA), Curve Agents (CA) and Surface Agents (SA). The number of agents varies based on the runtime needs of the application. This dynamic set of agents represents the database of the modeling environment. The different agents in the set along with the MA are responsible for managing the database autonomously and independently from the rendering and visualization operations.

The system architecture is illustrated in Fig. 8. The different agents are described in detail in the following subsections.

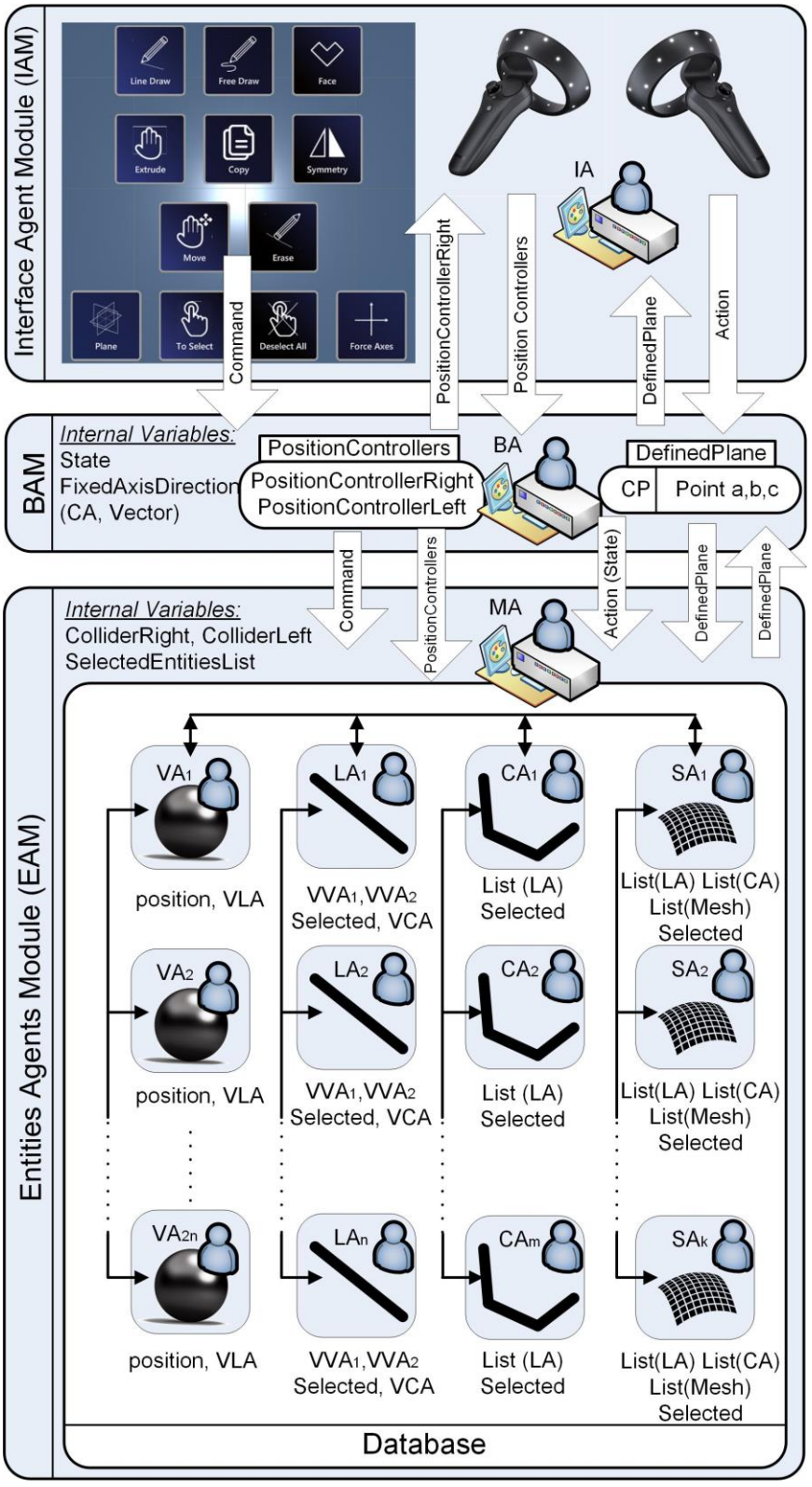


Fig. 8. Functional diagram of the agent-based architecture.

## *4.1 Interface Agent Module (IAM)*

The IAM is comprised of a single agent, the Interface Agent (IA), which is responsible for managing user interaction. The IA communicates with the BA by sending three types of messages: Command, Action, and Position Controllers. When the user uses the right controller to point to an item on the menu that is linked to the left controller and presses the trigger button, the IA sends a "command" message to the BA containing the specific command selected from the list shown in Fig 2. Alternatively, when the user presses or releases the trigger button of either controller without explicitly selecting a menu item, the IA sends an "action" message to the BA containing the specific user action: PressRight, ReleaseRight, or PressLeft. Finally, the IA sends "position controller" messages to the BA cyclically, which contain the position coordinates of the controllers.

The IA is also responsible for representing the work plane, if defined by the user, as well as the pointers of the controllers at the proper coordinates. To this end, a data structure in the BAM called DefinedPlane contains a counter (CP) from 0 to 4, which indicated the type of plane and a group of three points that define the work plane, if defined by the user. Otherwise, these values are null.

When the IA receives a message from the BA with the structure DefinedPlane, the IA acts as follows:

1. If the coordinates of all the points in DefinedPlaned are not null, the IA represents a work plane defined by these coordinates.
2. If the coordinates of all the points in DefinedPlaned are null, then:
    a. If the variable CP is 0, then the IA will delete all the planes.
    b. If the variable CP is 1, then the IA will represent a frontal plane with respect to the virtual scene which is linked to the left controller.
    c. If the variable CP is 2, then the IA will represent a profile plane with respect to the virtual scene which is linked to the pointer of the left controller.
    d. If the variable CP is 3, the IA will represent a horizontal plane with respect to the virtual scene which is linked to the pointer of the left controller.
3. If only some of the coordinates in DefinedPlane are null, (which may occur when CP = 4), the IA will represent a work plane defined by the coordinates of the points in DefinePlaned, the coordinates of the cursor that corresponds to the left controller, and if necessary, the coordinates of a vector perpendicular to the vector defined by the two previous points. This mechanism allows users to visualize a dynamic work plane that can adjust automatically based on the points that are introduced by interacting with the left controller.

Alternatively, if there is already a user-defined work plane, the IA will receive messages from the BA with the coordinates of the projection of the right controller on the plane. The IA then hides the cursor that represents the actual position of the right controller and shows a new cursor at the position indicated in the message sent by the BA. The actions performed by the IAM are illustrated in Fig. 9.

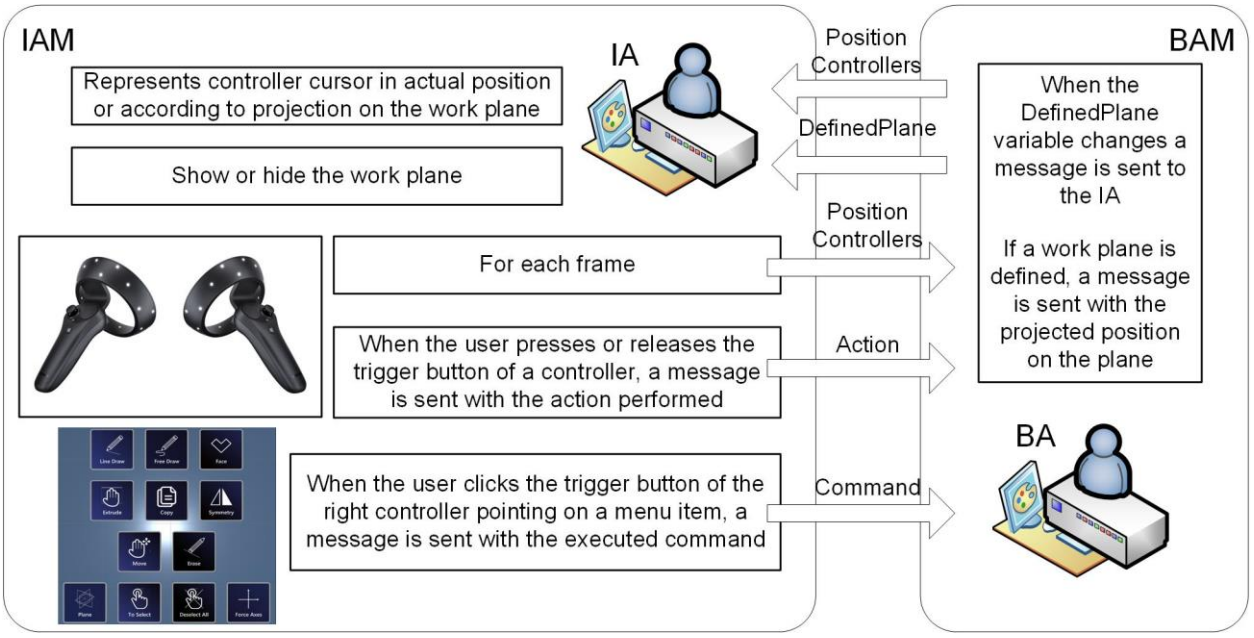


Fig. 9. Actions performed by an IAM.

## *4.2 Broker Agent Module (BAM)*

The BAM is comprised by a single broker agent (BA) which serves as a link between the IAM and the EAM. The following internal variables and structures are defined in the BAM:

1) State: it stores the update status of the selected command.
2) DefinedPlane: a structure that contains a counter (CP) with values from 0 to 4, which indicates the type of plane defined, and a group of coordinates that describe the points that define the plane, if user-defined. Otherwise, the values are null
3) FixedAxisDirection: structure comprised of a counter (CA) from 0 to 2, which indicates whether the cursor of the controller can move freely, parallel to the X-axis, or parallel to the Y-axis, and a vector that contains the direction for forcing the movement of the cursor, if user defined. Otherwise, the values are null.
4) PositionController: structure that stores the coordinates of the cursors of the left and right controllers in the variables PositionControllerLeft and PositionControllerRight, respectively.

Next, we analyze the behavior of the BA based on the type of message received from the IA.

### *4.2.1 Message type: Position controllers*

As discussed earlier, the IA repeatedly sends messages to the BA with the coordinates of the controllers' positions. When the BA receives one of these messages, it checks whether the set of points in the variable DefinedPlane are null to determine if there is a user-defined plane. If not null, the projection of the right controller on the plane needs to be calculated. Next, the BA chckes whether the FixedAxisDirection vector is null. If not, it calculates the coordinates of the previous projection over the direction defined by the FixedAxisDirection vector.

The resulting coordinates and the coordinates of the left controller are stored in the PositionControllers data structure. In the case of a user-defined plane, the BA sends a message to the IA with the PositionControllerRight variable, which is used by the IA to determine the position of the cursor of the corresponding controller and display it on the screen. In addition, if the "state" variable is StartLine, StartFree, StartMove, StartCopy, or StartExtrude, the PositionControllers variable is sent as a message to the MA.

The actions performed by the BAM when receiving a message of type position controllers are illustrated in Fig. 10.

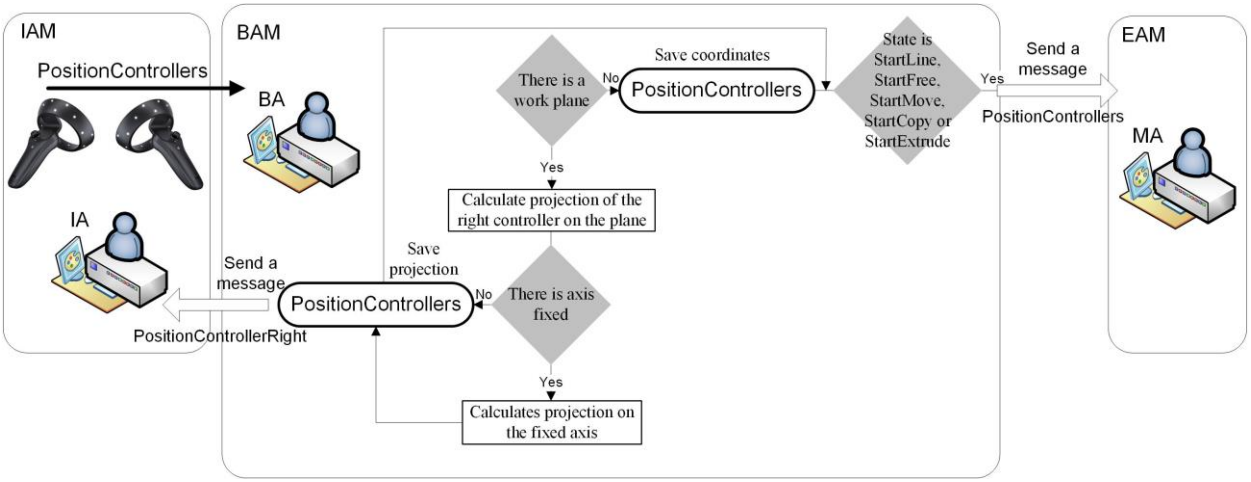


Fig. 10. Actions performed by the BAM when receiving a message of type "position controllers."

### *4.2.2 Message type: command*

When the user presses the trigger button of the right controller while pointing to a menu item (see Fig 2), the IA sends a message to the BA containing the specific command. When the BA receives this type of message and the value of the command is Line, Free, Extrude, Copy, Move, Erase, or Select, it updates the "state" variable to the value of the message. Otherwise, the following actions are performed:

1) If the message received is Face or Deselect, the BA sends a message to the MA with the command.
2) If the message received is Symmetry and all points in DefinedPlane are not null, the BA sends a message to the MA containing the command and DefinedPlane.
3) If the message received is Plane, the BA sets the coordinates of all the points in DefinedPlane to null and increases the value of the CP counter. Next, it checks whether CP is greater than 4. If so, the counter is reset to 0. Finally, it sends a message to the IA containing DefinedPlane.
4) If the message received is Axes and there is a defined plane (i.e. the coordinates of all the points in DefinedPlane are not null), the BA increases the value of the CA counter (and

resseting it to 0, if it is greater than 2). Next, the vector defined in FixedAxisDirection is set to null, if CA is zero; to the direction of the plane's X-axis, if CA equals one; and to the direction of the plane's Y-axis, if the value of CA is two.

The actions performed by the BAM when receiving a message of type command are illustrated in Fig. 11.

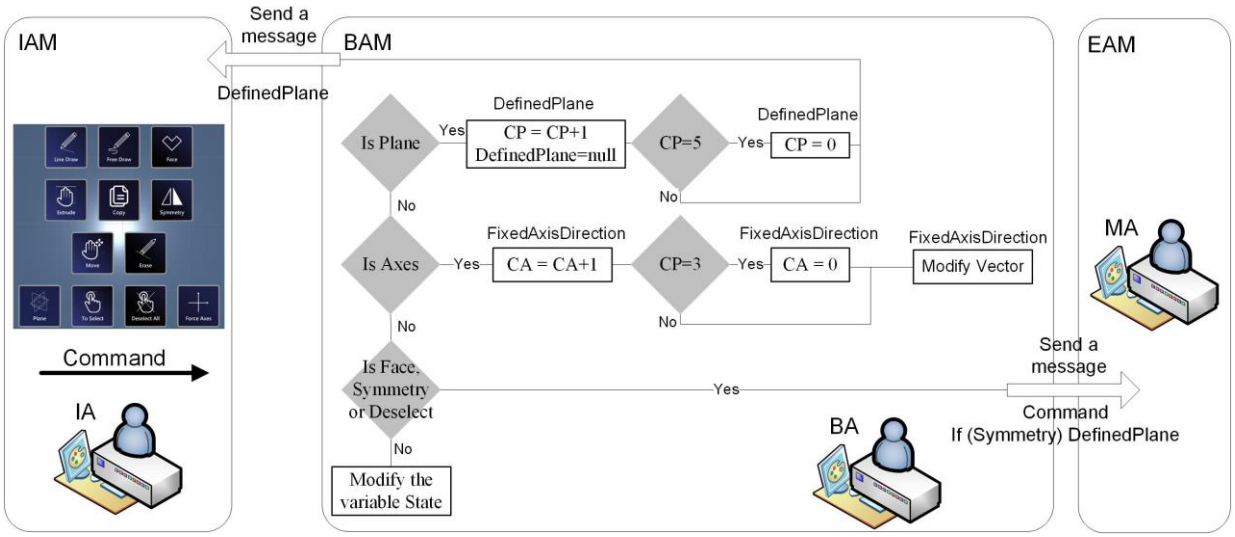


Fig. 11. Actions performed by the BAM when receiving a message of type "command."

### *4.2.3 Message type: action*

When the user presses the trigger button of a controller without specifically pointing to a menu item, the IA sends an Action type of message to the BA containing the action (i.e. PressRight, ReleaseRight, or PressLeft). Based on the value of the "state" variable and the type of action, the BA performs the following tasks:

1) If the content of the action message is PressRight:
    a. If the variable "State" is "Erase" or "Select," the BA sends a message to the MA with the variable "State."
    b. If the variable "State" is Line, Free, Extrude, Move, or Copy, the value is changed to StartLine, StarFree, StartExtrude, StartMove, or StartCopy, respectively, and a message is sent to the MA with the variables "State" and "PositionControllers."
2) If the content of the action message is ReleaseRight, and the variable "state" is StartLine, StartFree, StartExtrude, StartCopy, or StartMove, the value is changed to Line, Free, Extrude, Copy, or Move, respectively, and a message is sent to the MA with the variables "State" and "PositionControllers."
3) If the content of the action message is PressLeft, the counter CP of the variable DefinedPlane is not zero, and any of the points that define the plane is null, a message is sent to the MA with the variables "DefinedPlane" and "PositionControllers."

The MA will modify the coordinates of the points in DefinedPlane and reply to the message sent by the BA, which will then send a new message to the IA containing DefinedPlane. Finally, the IA will display the Work plane on the screen. The actions performed by the BAM when receiving a message of type "action" are illustrated in Fig. 12.

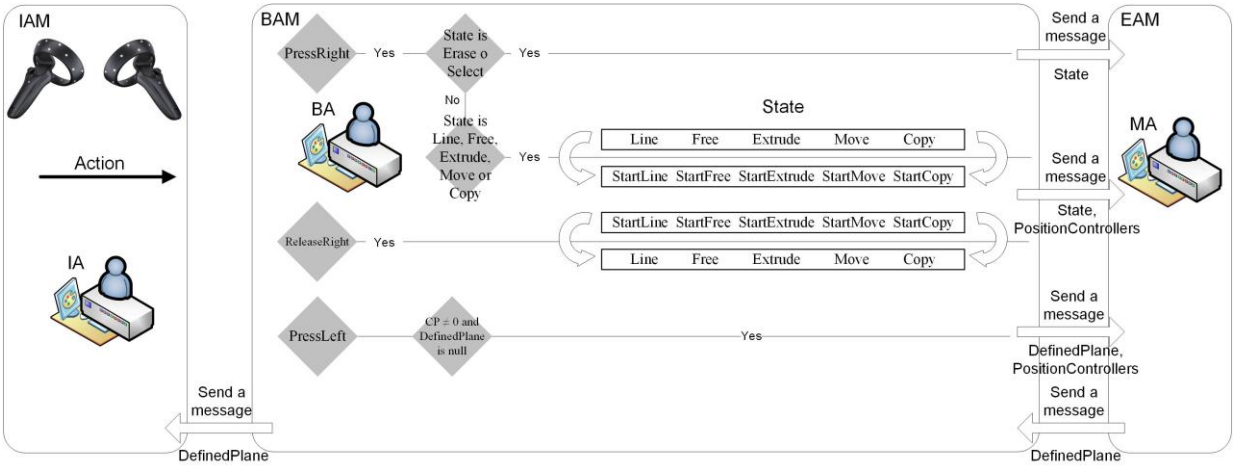


Fig. 12. Actions performed by the BAM when receiving a message of type "action."

## *4.3 Entities Agents Module (EAM)*

The EAM is comprised of a Manager Agent (MA) which coordintes the Vertex Agents (VA), Line Agents (LA), Curve Agents (CA), and Surface Agents (SA) that define the database of the application. The following variables are defined in the EAM:

1) ColliderRight and ColliderLeft: structures that record the name of the agent with which the left or right controller has collided as well as the coordinates of the collision point.
2) SelectedEntitiesList: list of agents in the EAM that have been previously selected by the user.

The different agents in the EAM are described in detail in the following subsections.

### *4.3.1 Vertex Agent (VA)*

A VA defines the starting or ending point in an LA. It is comprised of a GameObject of spherical form whose diameter is equal to the thickness of the lines used in the virtual space, making them visually unnoticeable. A VA is defined by its position (i.e. the spatial coordinates of its center point) and the variable VLA, which keeps track of the LA that it belongs to.

The VAs have been assigned the property Collider, provided by Unity for collision detection applications, in the form of "IsTrigger," We implement the methods "OnTriggerEnter" and "OnTriggerExit" to define the behavior of the agents. The representation of a VA is shown in Fig 13, left. The thin green line represents the spherical shape of the collider, whose diameter was established experimental.

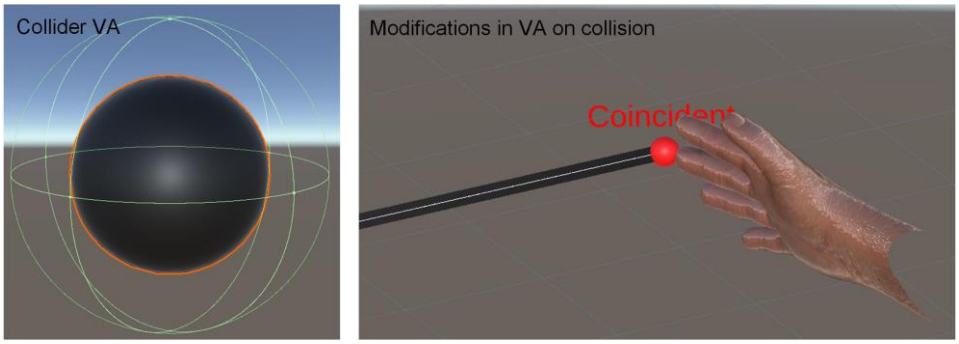


Fig. 13. Collider representation in a VA (left) and collision detected (right).

When a VA detects a collision with a controller, it gets resized, the color changes to red, and a "coincident" message is displayed (see Fig 13, right). Next, a message is sent to the corresponding MA indicating the controller involved in the collision and the coordinates of the position. When the VA detects the collision has ended, it returns the VA to its initial state and sends a message to the MA indicating the controller is no longer involved in a collision. The process is illustrated in Fig 14.

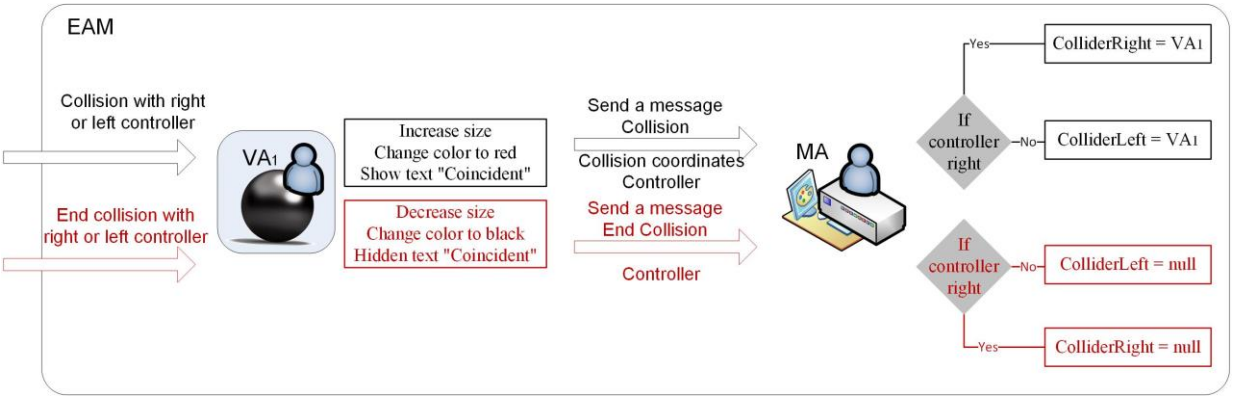


Fig. 14. Actions performed by the VA during a collision.

### *4.3.2 Line Agent (LA)*

An LA is comprised of an Empty GameObject, which has two scripts liked to it: SimpleLineDataProvider and Mixed Reality Line Renderer from Mixed Reality Toolkit (MRTK), to simulate the appearance of a solid object. The following variables are defined in an LA: $VVA_1$ and $VVA_2$, which contain the names of the VAs that define the starting and ending positions of the LA within the virtual scene; Selected, a Boolean variable that equals "true" when the agent is selected by the user, or "false" otherwise; and VCA, which keeps track of the CA that the LA belongs to, or null if the LA is not part of a CA.

Similar to the VAs, LAs also have an assigned Collider property, so when an LA detects a collision with the right controller, its color changes to red and a message is sent to the MA with information about the collision, the coordinates of the point of collision, and the value of VCA. When the collision ends, the LA returns to its initial state if the variable "Selected" is "false," or

changes to a blue color, if the value of the variable is "true." Once again, a new message is sent to the MA indicating the end of the collision and the value of VCA. The process is illustrated in Fig 15.

A close-up view of the end point of an LA is shown in Fig 16, left. The box and sphere colliders that correspond to the LA and VA, respectively, are shown in green. The behavior of an LA during a collision with the right controller is illustrated in Fig. 16, right.

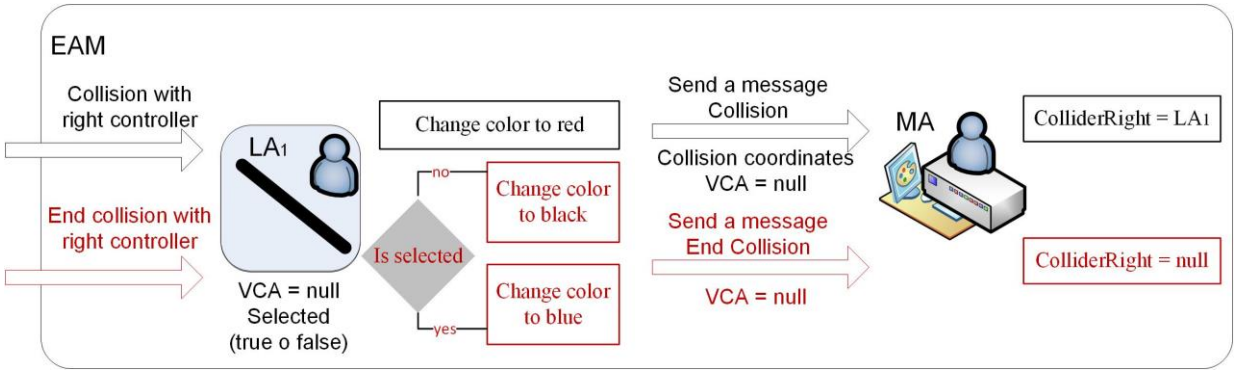


Fig. 15. Actions performed by the LA during a collision.

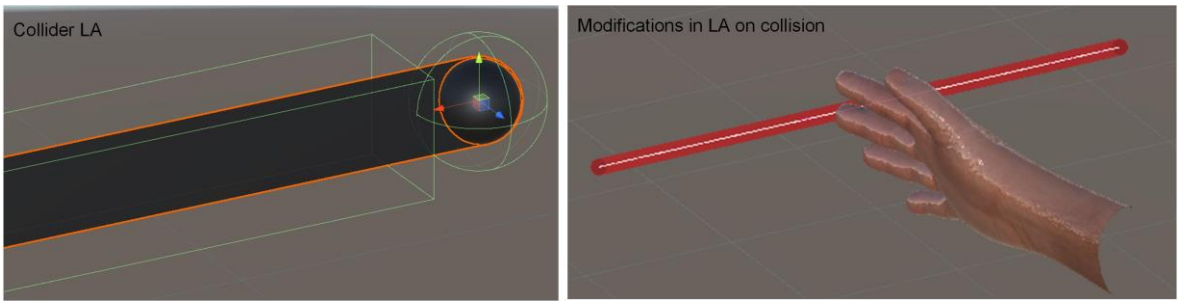


Fig. 16. Box and sphere colliders in an LA and VA (left) and behavior of an LA during a collision (right).

### *4.3.3 Curve Agent (CA)*

A CA is comprised of a list of consecutive LAs stored as a variable List(LA) where the second endpoint of each LA is coincident with the first endpoint of the next. All LAs use the variable VCA to keep track of the CA they belong to. When an LA is part of a CA, its behavior is conditioned by the rest of the LAs within that CA. Therefore, when an LA is involved in a collision with the right controller and sends the corresponding message to the MA with the collision coordinates and the VCA variable, the MA sends a new message to the CA listed in the VCA variable indicating the start or end of the collision. When a CA receives the message from the MA (or an SA, as we will discuss later), the CA notifies all the LAs listed in the variable List(LA) to change their colors to red (if the message was sent in response to a collision with the right controller) or black or blue, depending on whether the value of the variable "selected" is true or false. The process is illustrated in Fig 17.

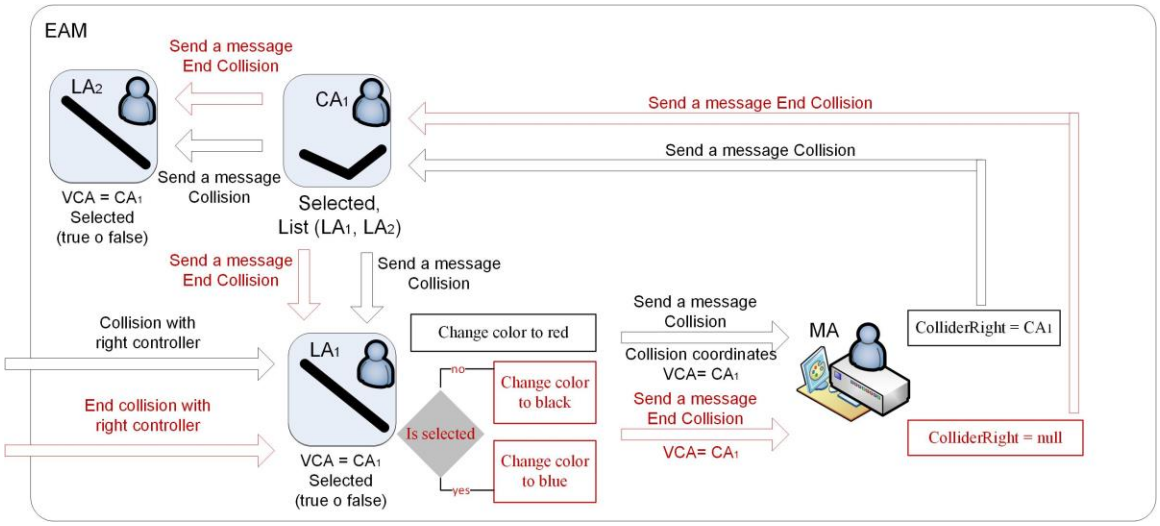


Fig. 17. Actions performed by the CA during a collision.

### *4.3.4 Surface Agent (SA)*

An SA is comprised of a list of meshes stored in the variable List (Mesh), and a list of LA and CA stored in List (LA) and List (CA), respectively, which delimit the contours of the meshes. When an SA is created, a Collider property is added to all the meshes, so they can all be responsive to collisions with the controllers.

When a collision (o lack thereof) between a mesh in an SA and the right controller is detected, the SA changes the appearance of all the meshes in the variable List (Mesh) and sends a message to all LAs and CAs that are part of List(LA) and List(CA). Finally, a message is sent to the MA with the collision coordinates and status, which is used by the MA to update the ColliderRight variable. The process is illustrated in Fig 18.

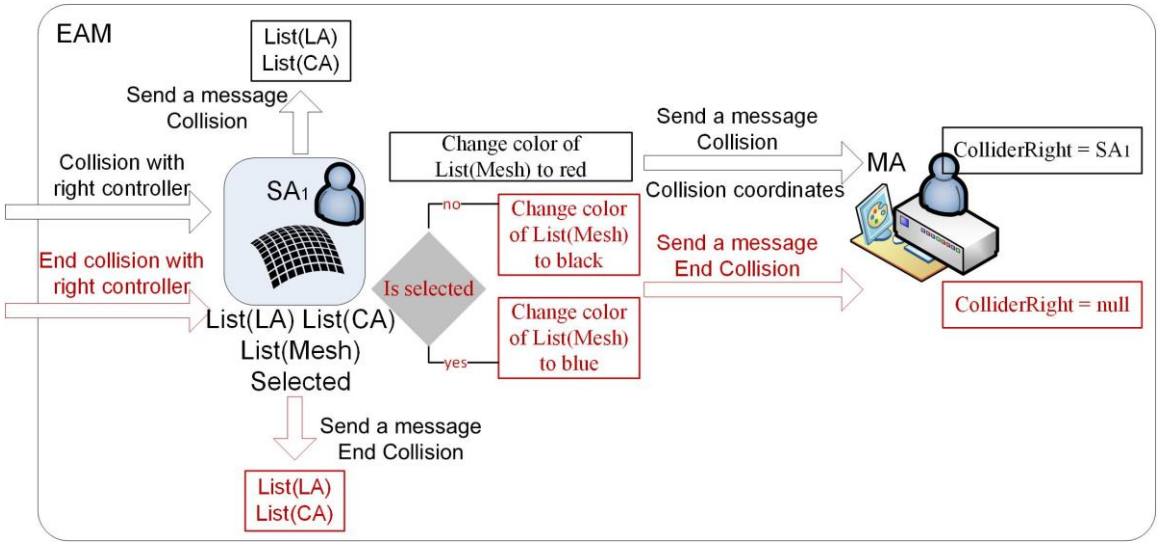


Fig. 18. Actions performed by the SA during a collision.

### *4.3.5 Manager Agent (MA)*

The MA is responsible for coordinating all the agents that are part of the EAM and for receiving messages from agents in the EAM and the BA. Next, we analyze the behavior of the MA.

- Behavior based on messages from the EAM

The messages received by the MA and sent by the EAM agents are triggered by collisions between these agents and the controllers. When the MA receives a message from a VA, LA, or SA indicating there has been a collision, the MA keeps track of the agent that sent the message and the collision coordinates in the structure ColliderRight or ColliderLeft, based on the controller that was involved in the collision. When the MA receives a message about the end of a collision, the variables ColliderRight or ColliderLeft are set to null. If the MA receives a message about a collision with an LA in which the VCA variable is not null, then the name of the agent and the collision coordinates are saved in the variable ColliderRight.

- Behavior based on messages of type "Action"

The MA implements a function to set the value of any variable. If the variable ColliderRight is a VA, the MA sets the coordinates of that VA. If ColliderRight is an LA, CA, or SA, the MA sets the collision cordinates sent by those agents to ColliderRight. If ColliderRight is null, the MA sets the coordinates sent by the BA to PositionControllerRight.

As discussed previously, when a user presses the trigger button of the right controller without pointing to specific menu item, the IA sends a message of type "action" with the content "PressRight" to the BA, which modifies the variable "State" and sends a message to the MA containing the variables "State" and "PositionControllers." Depending on the value of the variable "State," the MA performs the following tasks:

1. If the "State" variable equals StartLine, the MA creates two new VAs by setting their coordinates and linking them to a new LA through the variables VLA, $VVA_1$, and $VVA_2$.

   As long as the user is holding the trigger button, the BA sends messages to the MA with the variable PositionControllers and the MA modifies the coordinates of the second VA. The process repeats until the trigger button of the right controller is released, which forces the IA to send a message of type "action" with the content ReleaseRight to the BA, which in turn modifies the value of the variable "State" to "Line."

2. If the variable "State" equals StartFree, the MA creates two new VAs by setting their coordinates and linking them to a new LA through the variable VLA, $VVA_1$, and $VVA_2$. Next, it creates a new CA which is linked to the new LA by adding it to List(LA) and modifying the variable VCA in the LA.

   As long as the user is holding the trigger buton, the MA receives messages from the BA with the variable PositionControllers and sets the value of $VVA_2$ while the distance between $VVA_1$ and $VVA_2$ is less than a specific value determined experimentally. If the distance between $VVA_1$ and $VVA_2$ is greater than this value, the MA creates a new LA and the corresponding VAs, whose coordinates are the value of $VVA_2$ in the previous LA and the new LA is linked to the CA by adding it to List(LA) and modifying the variable VCA. The process is repeated for each new LA until the trigger button of the right controller is released, which forces the IA to send a message of type "action" with the

content ReleaseRight to the BA, which in turn modifies the value of the variable “State” to Free.

3. If the variable “State” equals StartMove and SelectedEntitiesList is not empty, the MA saves the coordinates to an auxiliary variable which will be used as a reference for the translation. As long as the user is holding the trigger button, the MA receives messages from the BA with the variable PositionControllers which is used by the MA to determine the translation of all the agents in SelectedEntitiesList. When the trigger button of the right controller is released, the IA sends a message of type “action” with the content ReleaseRight to the BA, which modifies the value of the variable “State” to Move.
4. If the variable “State” equals StartCopy and SelectedEntitiesList is not empty, the MA makes a copy of all the agents in SelectEntitiesList and applies the steps described in the previous point.
5. If the variable State equals Erase, the MA deletes the agent that is colliding with the right controller. Deleting an LA involves the deletion of the corresponding agents $VVA_1$ and $VVA_2$. Deleting a CA involves deleting all the LAs that are in its List(LA). If the variable ColliderRight equals an LA, the MA deletes the LA, which in tun will delete the agents $VVA_1$ and $VVA_2$. If ColliderRight is a CA, the MA will delete the CA, which in turn will delete all the LAs in its List(LA). Finally, if ColliderRight is an SA, the MA will delete the SA, which in turn will delete all the meshes in its List(Mesh), as well as all CAs in its List(CA) and all LAs in its List(LA). Every time an agent is to be deleted, the system checks whether the agent is in SelectedEntitiesList. If so, it is removed from the list.
6. If the variable “State” equals Select, the MA adds or removes the LA, CA, or SA that is colliding with the right controller from SelectedEntitiesList. Therefore, if ColliderRight is not null, the MA checks whether the agent exists in ColliderRight and in SelectedEntitiesList. If so, it is removed, and the variable “Selected” in the agent is changed to false. Otherwise, the variable “Selected” is changed to true and the agent is added to SelectedEntitiesList.
7. If the variable “State” equals StartExtrude and ColliderRight is not null, the MA will create a new SA whose geometry will depend on the agent referenced by ColliderRight, as follows:

   If ColliderRight references an LA or a CA, the agent is added to List(LA) or List(CA) of the newly created SA. Next, as long as the user is holding the trigger button and moves the right controller within a predefined distance, the MA creates a new agent of the same type as the one in ColliderRight, adds it again to List(LA) or List(CA), and places it at a distance from the previous one that is equal to the displacement performed by the right controller. The SA then creates the meshes defined between two consecutive LAs (in the case of extruding an LA) or between each consecutive pair of LAs and CAs. The extrusion process ends when the trigger button of the right controller is released. The MA creates two new CAs which will define the contour curves of the surface and will be added to List(CA) of the surface (see Fig. 19).

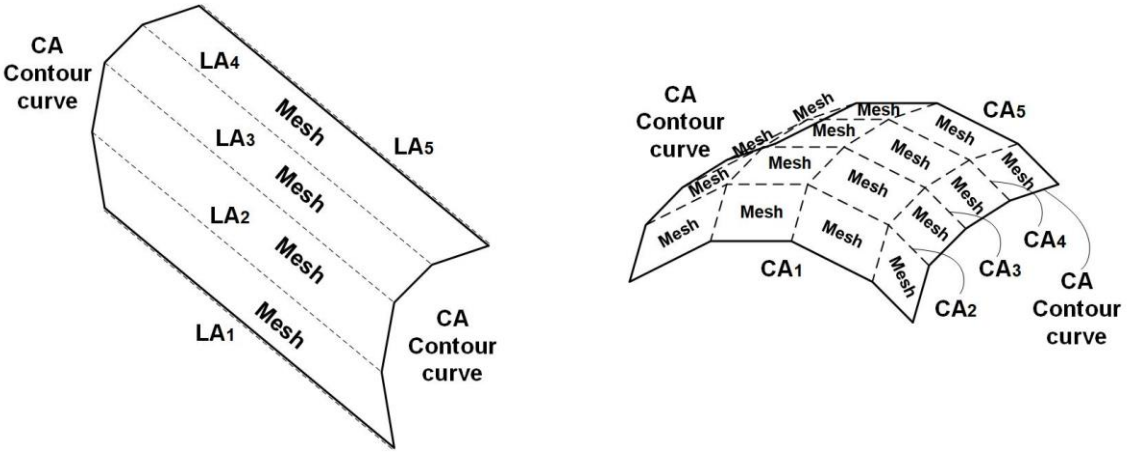


Fig.19. Surfaces generated by an extrusion process.

If ColliderRight references a VA at the beginning of the extrusion process, the MA creates a new SA and adds the LA referenced by its VLA to List(LA), if VCA is null. Otherwise, the CA that is in VCA is added to List(CA). Next, as long as the user is holding the trigger button, and moves the right controller within a predefined distance, the MA will create new agents which will be added to List(LA) or List(CA), and the SA will create a mesh which will be added to List(Mesh).

The extrusion process ends when the trigger button of the right controller is released. The MA creates two new CAs which will define the contour curves of the surface and will be added to List(CA) of the surface (see Fig. 20).

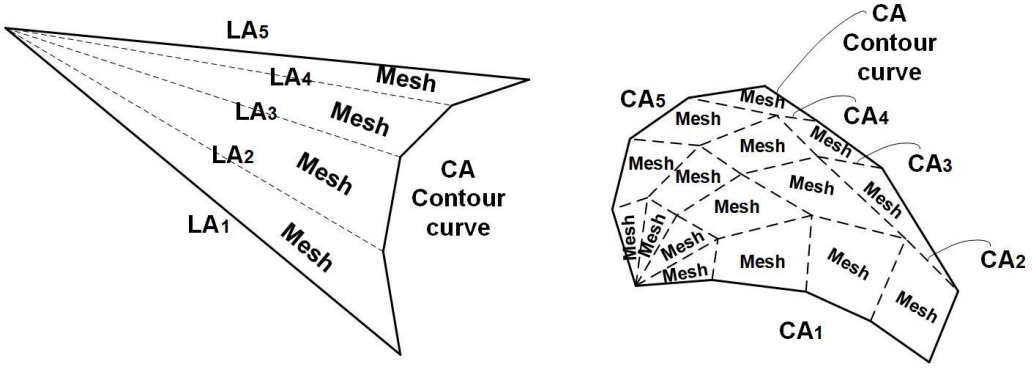


Fig.20. Surfaces generated by an extrusion process of a vertex.

Finally, if ColliderRight references an SA at the beginning of the extrusion process, the MA creates a copy of the existing SA which will be displaced according to the movement performed by the right controller while creating as many new SAs as agents define the contour of the SA being extruded.

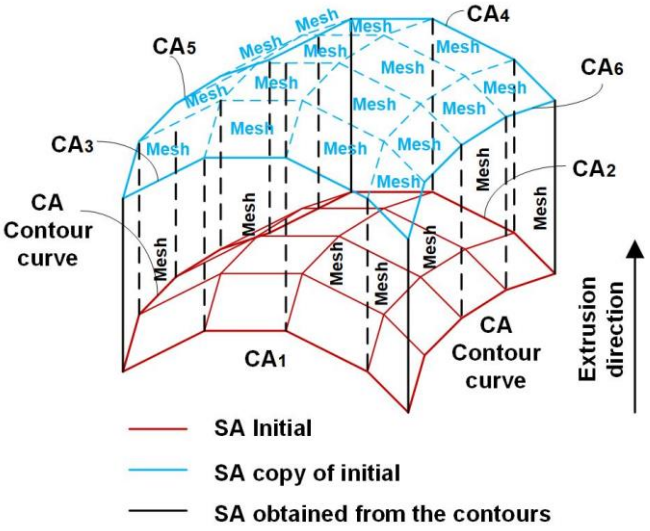


Fig.21. Surfaces generated by extruding a surface.

When the BA receives a message of type "action" containing PressLeft from the IA, if the CP counter of the variable DefinedPlane is not zero and any of the points that define the plane is null, the BA sends a message to the MA with the variables DefinedPlane and PositionControllers.

If the variable CP equals 4, the MA assigns coordinates to the first point that is null in the variable DefinePlane. Otherwise, the MA assigns coordinates to the first point in the variable DefinePlane and continues as follows:

1. If the variable CP equals 1, the coordinates of the other two points in the variable DefinedPlane are calculated so that the result is a frontal plane with respect to the orientation of the virtual environment.
2. If the variable CP equals 2, the coordinates of the other two points in the variable DefinedPlane are calculated so that the result is a profile plane with respect to the orientation of the virtual environment.
3. If the variable CP equals 3, the coordinates of the other two points in the variable DefinedPlane are calculated so that the result is a horizontal plane with respect to the orientation of the virtual environment.

Finally, the MA replies to the message sent by the BA returning the variable DefinedPlane.

- Behavior based on messages of type "Command"

As discussed previously, when the BA receives a message of the type "command" from the IA with the values Face, Symmetry, or Deselect, the BA sends a new message to the MA with the content of the original message and the variable DefinedPlane. The behavior of the MA for each of these messages is described next.

1. If the content of the message is "Face," the MA analyzes the contour defined by the elements stored in SeletedEntitiesList. If a surface can be defined, a new SA is created by

adding the entities from SelectedEntitiesList to List(LA) and List(CA). Next, the meshes that define the surface are created and added to List(mesh) of the newly created SA.

2. If the content of the message is “Symmetry” plus the variable DefinedPlane, the MA will create as many new agents as indicated by the SelectedEntitiesList. The agents Will also be assigned symmetrical coordinates with respect to the work plane defined by the points in DefinedPlane.
3. If the content of the message is “Deselect,” then for each agent in SelectedEntitiesList, the MA will change the value of the “Selected” variable to false, delete it from SelectedEntitiesList, and send a message to the agent so it can change its color to indicate it is not selected.

# 5 Experimental work

The implementation of our proposed architecture is built on the Unity platform using the behaviors that derive directly from the MonoBehaviour class as well as the messaging system associated to the GameObject class. To comparatively evaluate the concurrent functionalities of the proposed multi-agent algorithms versus a more traditional approach, we implemented a second version of the application in which all the algorithms run in a sequential manner for each rendered frame. In this version of our system, the detection of endpoints and points on geometric entities is accomplished analytically, without the use of Collider elements. Therefore, geometry detection and database management tasks are executed every frame before the virtual scene is refreshed.

For our evaluation, we used the set of fifteen 3D models shown in Fig 22, which were built using the two versions of the application, and implemented a mechanism in the algorithms to measure the time between two consecutive renders of the scene. The time between renderings for each model was then compared to the performance of the multi-agent system.

Our experiment was conducted in a Windows environment. Since there are certain Windows processes that cannot be stopped, the rendering times for a particular model may vary between measurements. Therefore, we considered the average time of a group of 60 measurements (one measurement per second for 60 seconds). We note that for the multi-agent version of our system, we can consider the times to be independent from the 3D model, since the rendering operation is independent from any other tasks in the application. The results of our study (shown in Fig 23) reveal that the multi-agent architecture performs significantly better in terms of timing between renders. Depending on the model, the time ranges between 9.63 and 17.12 milliseconds, which translates into a 36.87% to 50.94 % overall improvement.

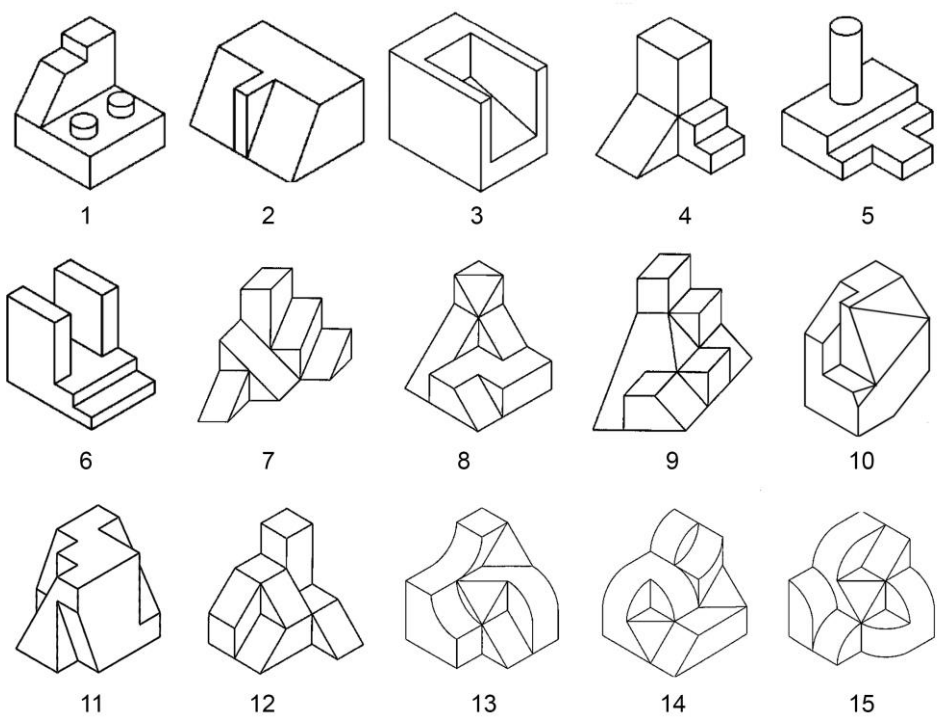


Fig. 22. 3D models used in our experiment.

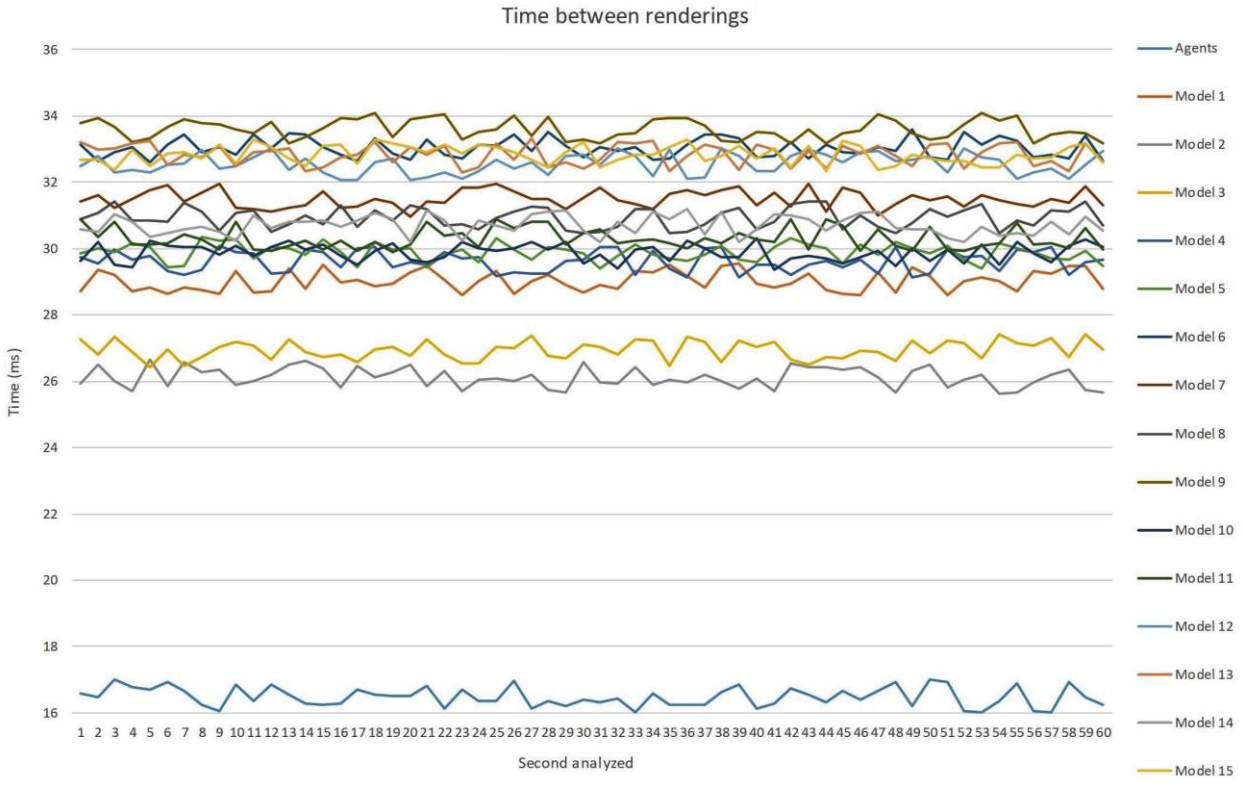


Fig 23. Times between renderings for the models analyzed.

When processed using traditional rendering techniques, the computational cost for each model depends on the calculations necessary to determine the proximity of the controllers to the geometric entities that are displayed. Therefore, for each vertex, the distances between the vertex and each of the two controllers must be calculated. Likewise, for each edge, the distance between the edge and each of the controllers must be calculated. Consequently, the computational cost for each model increases as the geometric complexity, in terms of number of vertices and edges, also increases, as shown in Fig. 24.

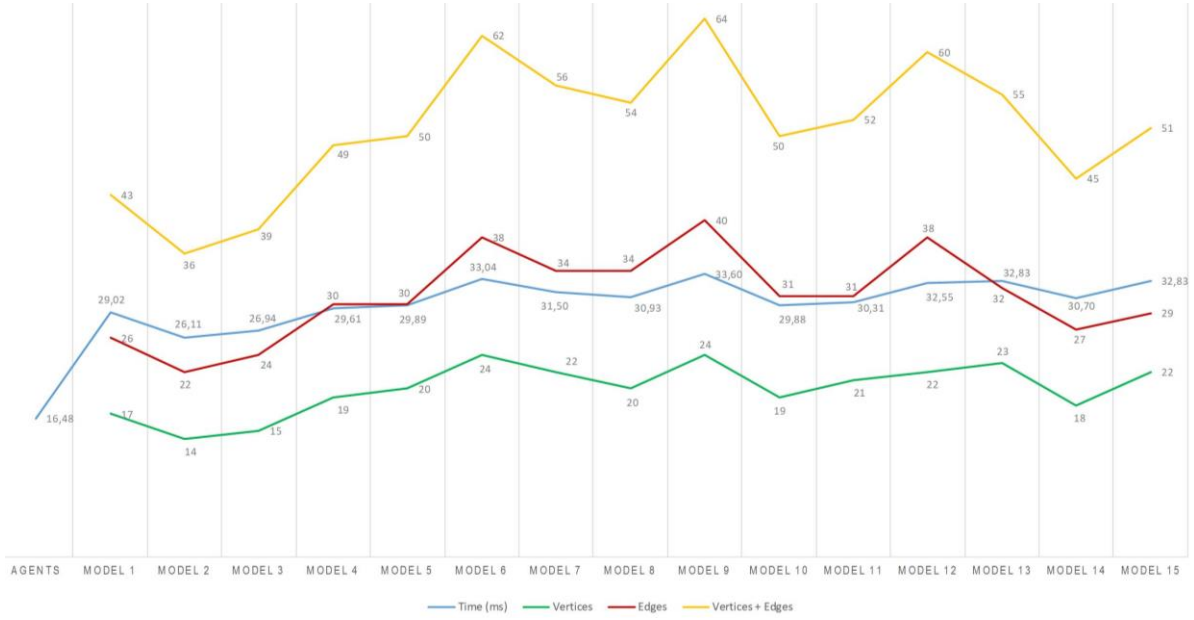


Fig 24. Computational cost by model based on geometric complexity

# 6 Conclusions

In this paper, we proposed a new 3D modeling paradigm for virtual environments where the detection of vertices, and points on entities, as well as the management of the geometric database, are performed independently from the rest of the application processes. We demonstrated a multi-agent architecture that can run different algorithms in a concurrent and independent manner, which enables the separation of rendering tasks from other types of processing in real-time applications and improves the graphical representation of and interaction with 3D models in virtual environments. Our contribution significantly reduces the computational load of both the CPU and GPU.

The architecture is structured in three modules: 1) management of the user interface, 2) a module that bridges the user interface with the application database, and 3) a module that independently manages the database. This de-centralized agent-based structure makes the overall architecture scalable and flexible.

We evaluated our system with a set of fifteen 3D models of varying geometric complexity defined as number of vertices and edges. Our comparative analyses showed that the multi-agent

approach significantly increases the rendering frequency of the scene between 36.87% and 50.94% when compared to sequential programming methods.

As future work, we are interested in expanding our performance evaluation to more complex models. We speculate that the differences would be even more significant when processing more complex geometry such as models with a complexity that is orders of magnitude higher (e.g. hundreds or thousands of vertices and edges) than our original dataset, or scenes with large numbers of models, such as mechanical assemblies and digital mock-ups of engineering designs. In addition, we are also interested in adapting our approach to multi-user interactive virtual environments.

## References


Arangarasan, R. y Gadh, R. (2000). "Geometric modeling and collaborative design in a multi-modal multi-sensory virtual environment." Proceedings of DETC'00 ASME 2000 Design Engineering Technical Conferences and Computers and Information in Engineering Conference Baltimore, Maryland, 1-19.

Arora, R., Kazi, R. H., Anderson, F., Grossman, T., Singh, K., & Fitzmaurice, G. W. (2017, May). Experimental Evaluation of Sketching on Surfaces in VR. In CHI (Vol. 17, pp. 5643-5654).

Bannova, O., Camba, J. D., & Bishop, S. (2019). Projection-based visualization technology and its design implications in space habitats. Acta Astronautica, 160, 310-316.

Bassanino, M., Wu, K. C., Yao, J., Khosrowshahi, F., Fernando, T., & Skjærbæk, J. (2010, July). The impact of immersive virtual reality on visualisation for a design review in construction. In 2010 14th International Conference Information Visualisation (pp. 585-589). IEEE.

Bleiweiss, A. (2009, February). Multi agent navigation on the GPU. In Games Development Conference (pp. 39-42).

Camba, J.D., Soler, J.L., & Contero, M. Immersive visualization technologies to facilitate multidisciplinary design education. In International Conference on Learning and Collaboration Technologies, pp. 3-11. Springer, Cham (2017).

Cappello, F., Ingrassia, T. y Romano, M. (2007). "VR Studio: Solid Modelling in a Virtual Reality Environment." 5th Eurographics Italian Chapter Conference 2007 – Proceedings, 119-126, 1.

Chen, Q., Luo, G., Tong, Y., Jin, X., & Deng, Z. (2019). Shape- constrained flying insects animation. Computer Animation and Virtual Worlds, 30(3-4), e1902.

Chevaillier, P., Harrouet, F., Reignier, P., & Tisseau, J. (2000). Virtual reality and multi-agent systems for manufacturing system interactive prototyping. International Journal of Design and Innovation Research, 2(1), 90-101.

Chu, C., Dani T., and Gadh R. (1997). "Multi-sensory user interface for a virtual-reality-based computer-aided design system." CAD Computer Aided Design, 709-725, 29(10).

Cordeiro, E., Giannini, F. y Monti, M. (2019). "A survey of immersive systems for shape

manipulation". Computer-Aided Design and Applications, 1146-1157, 16(6).

Cruz-Neira, C., Sandin, D. J., & DeFanti, T. A. (1993, September). Surround-screen projection-based virtual reality: the design and implementation of the CAVE. In Proceedings of the 20th annual conference on Computer graphics and interactive techniques (pp. 135-142).

Eberly, D. (2008). Triangulation by ear clipping. Geometric Tools, 2002-2005.

Gonzalez-Morcillo, C., Weiss, G., Jimenez, L., Vallejo, D., & Albusac, J. (2007, September). A MultiAgent System for Physically Based Rendering Optimization. In International Workshop on Cooperative Information Agents (pp. 149-163). Springer, Berlin, Heidelberg.

Jerald, J. The VR book: Human-centered design for virtual reality. Morgan & Claypool, (2015).

Keefe, D. and Jackson, B. (2016). "Lift-Off: Using Reference Imagery and Freehand Sketching to Create 3D Models in VR". IEEE Transactions on Visualization and Computer Graphics, 1442-1451, 22(4).

Kolasinski, E. M. (1995). Simulator sickness in virtual environments (Vol. 1027). US Army Research Institute for the Behavioral and Social Sciences.

Ladwig, P., Herder, J. y Geige, C. (2017). "Towards Precise, Fast and Comfortable Immersive Polygon Mesh Modelling: Capitalising the Results of Past Research and Analysing the Needs of Professionals." International Conference on Artificial Reality and Telexistence and Eurographics Symposium on Virtual Environments.

Lipman, R., and Reed, K. Using VRML in construction industry applications. Web3D –VRML 2000 Symposium, Monterey, CA, 1-7 (2000).

Machuca, M. D. B., Asente, P., Stuerzlinger, W., Lu, J., & Kim, B. (2018, October). Multiplanes: Assisted freehand VR sketching. In Proceedings of the Symposium on Spatial User Interaction (pp. 36-47).

McGraw, T., Garcia, E. y Sumner, D. (2017). "Interactive swept surface modeling in virtual reality with motion-tracked controllers". Proceedings – Sketch-Based Interfaces and Modeling, Part of Expressive 2017, (1).

Messner, J.I., Yerrapathruni, S.C., Baratta, A.J., and Riley, D.R. Cost and schedule reduction of nuclear power plant construction using 4D CAD and immersive display technologies. Computing in Civil Engineering: Proceedings of the International Workshop of Information Technology in Civil Engineering, 136-144 (2002).

Monteiro, P., Coelho, H., Gonçalves, G., Melo, M., & Bessa, M. (2019). Comparison of Radial and Panel Menus in Virtual Reality. IEEE Access, 7, 116370-116379.

Nisha, B. (2019). The pedagogic value of learning design with virtual reality. Educational Psychology, 39(10), 1233-1254.

Oliveira, E., Fischer, K., & Stepankova, O. (1999). Multi-agent systems: which research for which applications. Robotics and Autonomous Systems, 27(1-2), 91-106.

Opdenbosch, A., and Hastak, M. Virtual reality environment for design and analysis of automated construction equipment. 2nd Congress for Computing in Civil Engineering, Atlanta, GA, 566-573 (1994).

Rangel-Kuoppa, R., Avilés-Cruz, C., & Mould, D. (2003, September). Distributed 3d rendering system in a multi-agent platform. In Proceedings of the Fourth Mexican International Conference on Computer Science, 2003. ENC 2003. (pp. 168-175). IEEE.

Rosales, E., Rodriguez, J. y Sheffer, A. (2019). "Surfacebrush: From virtual reality drawings to manifold surfaces." ACM Transactions on Graphics, 38(4).

Ryken, M. J., & Vance, J. M. (2000). Applying virtual reality techniques to the interactive stress analysis of a tractor lift arm. Finite elements in analysis and design, 35(2), 141-155.

Schlechtweg, S., Germer, T., Strothotte, T.: Renderbots multiagent systems for direct image generation. Computer Graphics Forum 24, 137–148 (2005).

Schmitt, G. Virtual Reality in Architecture. In N. M. Thalmann and D. Thalmann (Eds.), Virtual Worlds and Multimedia. Chichester, England: Wiley (1993).

Sharma, S., Jerripothula, S., Mackey, S., & Soumare, O. (2014, December). Immersive virtual reality environment of a subway evacuation on a cloud for disaster preparedness and response training. In 2014 IEEE symposium on computational intelligence for human-like intelligence (CIHLI) (pp. 1-6). IEEE.

Shibano, N., Hareesh, P.V., Kashiwagi, M., Sawada, K., and Takemura, H. Development of VR experiencing system with hemi-spherical immersive projection display for urban environment design. Proceedings of the Seventh International Conference on Virtual Systems and Multimedia (VSMM'01) (2001).

Soler-Dominguez, J. L., De-Juan-Ripoll, C., Camba, J. D., Contero, M., & Alcañiz, M. (2019, July). Gaming Background Influence on VR Performance and Comfort: A Study Using Different Navigation Metaphors. In International Conference on Human-Computer Interaction (pp. 646-656). Springer, Cham.

Tessier, C., Chaudron, L., and Muller, H.J. Conflicting Agents: Conflict Management in Multi-Agent Systems. Kluwer Academic Publishers, Boston, 2002.

Trika, S.N., Banerjeet, P. y Kashyap, R.L. (1997) "Virtual reality interfaces for feature-based computer-aided design systems." Computer-Aided Design, 565-5774, 29(8).

Wang, Y., Dubey, R., Magnenat-Thalmann, N., & Thalmann, D. (2013). An immersive multi-agent system for interactive applications. The Visual Computer, 29(5), 323-332.

Whisker, V.E., Baratta, A.J., Yerrapathruni, S., Messner, J.I., Shaw, T.S., Warren, M.E., Rotthoff, E.S., Winters, J.W., Clelland, J.A., and Johnson, F.T. Using immersive virtual environments to develop and visualize construction schedules for advanced nuclear power plants. In Proceedings of ICAPP, vol. 3, pp. 4-7. (2003).

Wooldridge, M (2002). An Introduction to MultiAgent Systems. John Wiley & Sons. p. 366. ISBN 978-0-471-49691-5.

Xiao, H., Li, J., & Deng, L. P. (2011, September). Designing of particle system architecture by using multi-agent modeling. In 2011 International Conference on Electrical and Control Engineering (pp. 4441-4444). IEEE.

Zhong, Y., Shirinzadeh, B. y Ma, W. (2005). "Solid modelling in a virtual reality envronment." The Visual Computer, 17-40, 21(1-2).

Zorriassatine, F., Wykes, C., Parkin, R., & Gindy, N. A survey of virtual prototyping techniques for mechanical product development. Proceedings of the institution of mechanical engineers, Part B: Journal of engineering manufacture, 217(4), 513-530 (2003).