Computers & Graphics (2019)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

# LifeBrush: painting, simulating, and visualizing dense biomolecular environments

Timothy Davison*, Faramarz Samavati**, Christian Jacob

*ICT 2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4*

ARTICLE INFO

*Article history*:
Received May 12, 2019

*Keywords:* Agent-based simulation, Discrete element texture synthesis, Sketch-based modelling, Virtual Reality, Visualization

ABSTRACT

LifeBrush is a Cyberworld for painting dynamic molecular illustrations in virtual reality (VR) that then come to life as interactive simulations. We designed our system for the biological mesoscale, a spatial scale where molecules inside cells interact to form larger structures and execute the functions of cellular life. We bring our immersive illustrations to life in VR using agent-based modelling and simulation. Our sketch-based brushes use discrete element texture synthesis to generate molecular-agents along the brush path derived from examples in a palette. In this article we add a new tool to sculpt the geometry of the environment and the molecules. We also introduce a new history based visualization that enables the user to interactively explore and distil, from the busy and chaotic mesoscale environment, the interactions between molecules that drive cellular processes. We demonstrate our system with a mitochondrion example.


## 1. Introduction

Biological systems span from whole organisms, down to the scale of viruses and individual molecules. At the mesoscale, molecules interact to form more complicated structure and function. The mesoscale mitochondrion is an internal compartment within the Eukaryotic cell that assembles adenosine-triphosphate (ATP) molecules, which are used by other cell components to do work (Figure 1). The mitochondrion is a dense and chaotic space, yet highly organised [1, 2]. A significant challenge for researchers has been communicating scientific findings at this level because visible light microscopes do not reveal the functional components at this scale.

Scientific illustrators have confronted the challenges posed by the mesoscale. For instance, David Goodsell [2] is famous for his painstakingly detailed 2D watercolour paintings of mesoscale environments—his illustration of the mitochondrion is the inspiration for Figure 1. Meanwhile, animating mesoscale scenes was a laborious process for a team of 3D animators in Harvard's Biovisions project [3]. Imagine if we could paint these illustrations in 3D space and then have them come to life around us, to explore and manipulate.

The dense and chaotic mesoscale environments of the cell pose some significant challenges: (1) how to fill and simulate an environment with a large number of molecules and (2) how to visualise the chaotic interactions between molecules. Manual and random placement, together with agent-based simulation in a video game engine, was employed for Prokaryotic and Eukaryotic simulations [4, 5]. Klein et al. [6] use the power of the GPU to create large and densely packed mesoscale environments through parameterisation automatically.

Inspired by Goodsell's 2D illustration work, we created *LifeBrush* [7] as a sketch-based Cyberworld to paint 3D mesoscale illustrations in virtual reality (VR) that one can step inside as they come to life (Figure 2). We propose novel visualisations to trace interactions between molecules as the simulation progresses through time. Our system couples interactive sketch-based design with an agent-based model of molecular interactions.

Agent-based modelling has been used to capture the swarm-

*Corresponding author: email: tbdaviso@ucalgary.ca
**email: samavati@ucalgary.ca
email: cjacob@ucalgary.ca

ing behaviour of birds [8] and for modelling molecules in biological simulations [4, 5]. We use agent-based modelling to abstract the behaviour and interaction of molecules from the underlying and expensive to compute quantum dynamics that govern those interactions, which is essential for a real-time VR environment. A molecular-agent in our system has a set of behaviours and attributes that determines how it interacts with its environment and other agents. We implemented an agent-based framework that is capable of simulating and rendering 10,000 agents at 90 frames-per-second in our mitochondrion example (Section 3).

The user creates example arrangements and configurations of agents in a palette (Figure 2). We paint molecules, derived from the arrangements in the palette, along the brush-path, in space, and on surfaces (Section 5). To generate the molecules, we use a discrete element texture synthesis algorithm, that we previously described in [9], that uses the palette as an example for synthesis. We propose a simple mapping to convert between our system's representation of an agent and the internal representation of a so-called element within the texture synthesis framework (Section 4). Switching back and forth between painting and simulating enables iterative design possibilities (Figure 3).

In this extended article on *LifeBrush* [7] we added a sculpting tool, based on implicit surface modelling [10]. It enables the user to quickly sketch the geometry of the environment and the rough shape of proteins (Figures 13 and 11). The sculpting tool acts like virtual clay; the user either adds or removes the virtual clay with the brush point (Section 6).

In our original *LifeBrush* mitochondrion [7], it was difficult to observe and understand when and how molecular-agents interact. In this extended article, we propose a novel historical visualisation of events within the VR environment, where the user queries a set of agents and interactions to see a trace of the spatial history and sequence of events that led to the interactions (Section 7).

The complete source code and examples used in this article are available under an MIT open source license (`https://github.com/timdecode/LifeBrush`). A video based on the figures in this article is available at `https://youtu.be/iOWU_LiCxKI`.

## 2. Related work

### 2.1. Molecular Dynamics Construction and Visualization

There are many different techniques for producing 3D visualizations of molecular and mesoscale structures [11, 12]. To create molecular scenes, Packmol [13] and CellPack [14] randomly pack proteins and molecules onto surfaces and regions inside of a virtual cell according to user-created recipe files. Klein et al. [6] accelerate the packing process with GPUs. Koch et al. [15] reduce visual clutter arising from ambient occlusion artefacts in 3D multi-scale visualizations of molecular scenes, while Kouřil et al. [16] address the problem of label placement in dense 3D molecular scenes. CellView [17] is an interactive visualization tool for exploring multi-scale visualizations of structures in the cell down to the molecular level. In *LifeBrush* [7], we introduced interactive sketch-based design and simulation for molecular scenes, within VR. This article addresses limitations of that work, with new visualization and 3D sculpting tools.

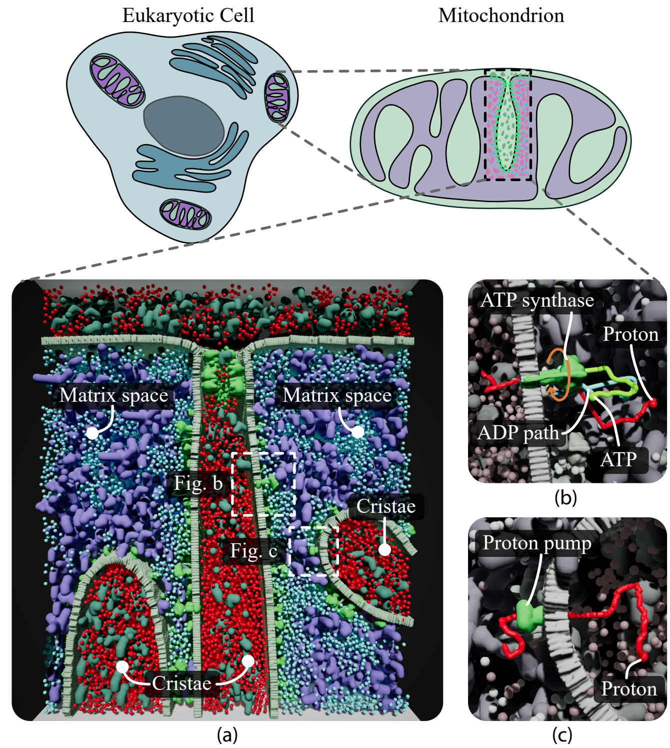



Fig. 1: The mitochondrion is an organelle in Eukaryotic cells. It generates most of the cell's adenosine triphosphate (ATP), a source of chemical energy. A proton gradient between the matrix space and cristae drives protons (in red) through ATP synthase (in green) enzymes in the inner mitochondrial membrane, causing the enzymes to spin. ATP synthase uses its kinetic energy to combine phosphate and adenosine diphosphate (ADP, in cyan) to produce ATP (green spheres). Hydrogen pumping proteins (in green) in the inner membrane move hydrogen from the matrix space to the cristae. To increase the number of ATP synthase enzymes the mitochondrion is packed with cristae, increasing its internal surface area and the rate of ATP synthesis. We sketched and simulated this mitochondrion, which contains 10,000 molecules, in VR using *LifeBrush*.

### 2.2. Agent-based modeling and visualization

Agent-based approaches have been used to model biological systems like swarming insects and birds [8], without relying on purely mathematical models [18]. Agent-based systems have also been used to model the Lactose operon inside *E. coli* bacteria [19], for gene regulation [20] and immune system models [21]. Along the lines of mathematical whole-cell models [22], agent-based models have been applied to both Prokaryotic [4] and Eukaryotic cells [5]. Meanwhile, multi-scale agent-based models can simultaneously capture cells and groups of cells at different scales [23, 24]. Automatic abstraction has been used to reduce the computational complexity of such models [25]. Pathline visualizations have been applied to swarm systems [26, 27]. We follow a similar idea for path visualization, but with the addition that the user can query events and agents from a historical simulation timeline in VR. We have also added a novel trace component to the visualization to trace dependencies between interactions.

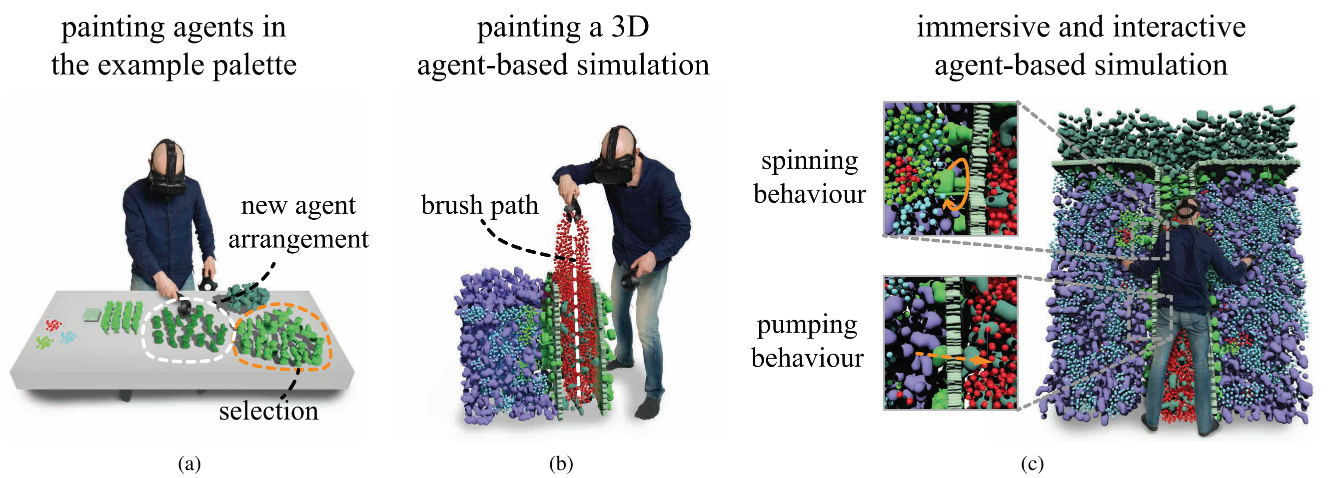



Fig. 2: Immersive design and simulation in VR. (a) Creating and configuring a new arrangement of molecular-agents (white outline) through a copy-paste operation from another set of molecular-agents (orange outline). (b) **Painting** the red agents from (a) into an agent-based simulation. (c) Stepping into an **immersive simulation**, the painted molecular-agents come to life. In the close-ups, molecules in the mitochondrion spin and pump red proton agents along the orange arrow.

### 2.3. *Example-based texture synthesis, procedural modelling, and sketch-based synthesis*

The goal of example-based texture synthesis algorithms is to create a large non-repeating output that is similar to an input texture [28]. Pixel-based approaches synthesize 2D textures [29, 30, 31, 32, 33, 34]. Multi-scale 2D texture synthesis allows textures with very high [35] or infinite resolution [36]. Discrete element textures manipulate individual discrete elements instead of pixels. For example, texture bombing splatters small texture elements into a larger texture [37] and more recently this has been optimized for on-the-fly generation on GPUs [38, 39]. Hurtut et al. [40] and Landes et al. [41] consider the shape of elements during synthesis. In this article, we use our interactive discrete element texture synthesis algorithm, with support for multiple textures in a palette, to generate agents [9]. This algorithm is related to the works of Ma et al. [42], Ijiri et al. [43] and Roveri et al. [44]. We use a simple mapping from agents to elements (Section 5), and as far as the authors are aware, our original work [7] is the first to apply example-based discrete element texture synthesis to the problem of generating an agent-based simulation.

Sketch-based interfaces apply the familiarity of real-world tools like pencil and paper to interactive design problems, such as 3D modelling [45]. Ecosystem simulation has been used to synthesize and render large plant ecosystems [46]. Discrete element texture synthesis has been combined with sketch-based synthesis to allow the user to guide the synthesis process [43, 47, 48, 44, 9]. Ketabchi et al. [49] and Samavati and Runions [50] apply interactive 3D content modelling to the digital earth project. Sketch-based interfaces have also been used to design and guide dynamic fluid simulations [51] and for sketching crowds of agents [52]. A limitation of 2D sketch-based interfaces is how to embed a 2D curve from the computer screen into a 3D space. The advent of consumer VR devices solves some of the problems with 2D sketch-based synthesis, by adding six degree-of-freedom input. For example, sketch-based interaction has been used in VR to paint curves in virtual space [53]. We build on discrete element synthesis and sketch-based interaction for creating and configuring molecular-agents in a 3D VR simulation.

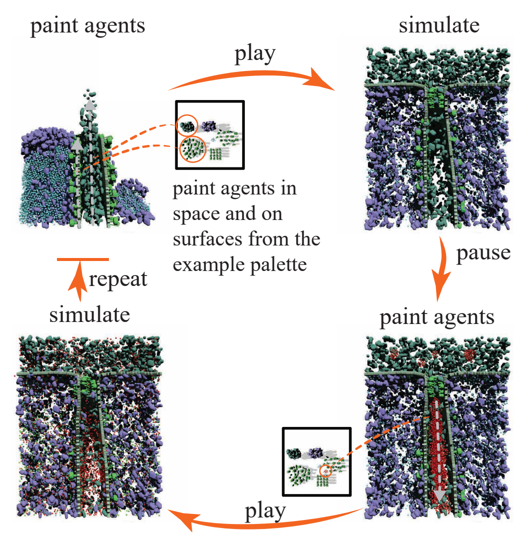



Fig. 3: An interactive exploration loop. We paint a simulation, press play and simulate. We pause, paint new agents and repeat. The example palette enables us to create new agent arrangements and configurations to paint into our simulations.

## 3. Large-scale agent-based simulation in Unreal Engine 4

An **agent** is defined by the set of *situations* that the agents can be in, its *actions*, its *internal data* and a *decision function* that determines what actions to take, given internal data and the current situation [54]. An agent interacts with other agents and its environment.

Our agent-based framework is built on the well known entity-component-system architecture [55], which we implemented within *LifeBrush*. In this architecture, we store the state of an agent in components attached to the agent entity. Systems implement agent behaviour by accessing and modifying the components attached to an agent. We integrated our implementation with the Unreal Editor so that users can utilize Unreal's 3D widgets, property editor interface, and serialization system [56].

For performance reasons, we chose to implement an entity-component-system (ECS) instead of using Unreal's actor-component model. The Unreal Engine is generalizable to a wide variety of games. However, that generalizability meant that we were not able to simulate more than a few hundred molecular-agents in real-time. Consequently, we carefully optimized our ECS implementation to store structures of the same type in contiguous blocks of memory. Systems enumerate the components of a given type, one after the other, enabling the processor to keep data in its fast CPU-caches without accessing its slow main memory. Efficient cache utilization and small size structures are the primary reason that we can achieve higher performance than the actor-component model that Unreal uses natively. The Unity game engine has recently released a similar entity-component-system architecture to our implementation and likewise, simulate a significant number of agents [57].

To render so many agents, we apply GPU instancing, an efficient technique that uses hardware features to reduce the number of draw calls necessary to render many objects with the same geometry and material properties. With so many agents, running rigid body physics calculations on the CPU is too expensive for a real-time simulation. Therefore, we integrated Nvidia's Flex GPU particle-physics engine with our entity-component-system [58, 59].

### 3.1. Mitochondrial molecular-agents

Swarm agents implement rules that determine how they behave when other agents fall within zones of interaction [8]. Following this model of swarm behaviour, we use interaction zone rules to govern the behaviour of our molecular agents (see Figure 4). We store the state of the agent behaviours in components, with the implementation of the rules defined in systems. Each agent has a Flex particle that determines its physical interaction with other agents and the environment [58]. Except for the molecular-agents on the mitochondrial surfaces, each agent also has random walk behaviour to model Brownian motion.

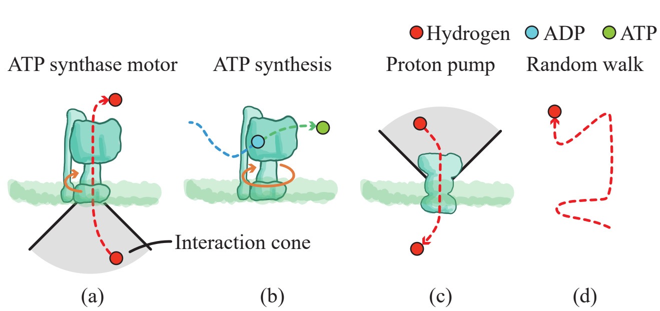



Fig. 4: Molecular-agent behaviours. (a) The ATP synthase motor behaviour causes ATP synthase to spin when a hydrogen agent enters its interaction cone. (b) ADP is converted into ATP by a spinning ATP synthase. (c) Protons are pumped from within the interaction cone of a proton pump to the other side of the membrane. (d) The random walk behaviour causes an agent to randomly change direction at random time intervals, simulating Brownian motion.

## 4. Synthesizing molecular-agents

### 4.1. Discrete element texture synthesis

*LifeBrush* uses a discrete element texture synthesis algorithm that we developed previously (see [9]) to generate molecular-agents. Agents are not elements in that system. A **discrete element** is a particle with a position, radius and an attribute vector to store user-defined attributes of the element. We use a map to convert back and forth between agents and elements (Figure 6).
A discrete element texture has the property that the arrangement of elements is locally similar in a small window to other regions of the texture [28]. Our algorithm generates agents so that the windows around those elements are similar to windows in the example [9]. The algorithm requires a measure for how similar the attribute vectors of two elements are.

In *LifeBrush*, the attribute vector consists of two components, an *appearance* identifier and a *behaviour* identifier. The vector components are set during the mapping from agent to element. The *appearance* identifier is a unique integer for the combination of mesh and material properties of an agent. If two agents have the same mesh and material, they will have the same *appearance* identifier. The *behaviour* identifier is also an integer for the unique combination of components attached to an agent. If two agents have the same set of components, they will have the same *behaviour* identifier.

Let $a$ and $b$ be elements, with attribute vectors $[\alpha_a, \beta_a]$ and $[\alpha_b, \beta_b]$ respectively, where $\alpha_a$ is the appearance identifier for $a$ and $\beta_a$ is its behaviour identifier. The similarity measure between $a$ and $b$ is given by:

$$|a - b| = \omega_0(\alpha_a, \alpha_b) + \omega_1(\beta_a, \beta_b), \tag{1}$$

where $\omega_0$ and $\omega_1$ are customizable functions to compare two attributes. In our implementation, $\omega_0$ and $\omega_1$ are 0 when their parameters are the same and 1 when not. Our discrete element texture synthesis algorithm uses the element similarity measure to match elements that are the same (by appearance and behaviour) in the output and example [9]. With two attribute components, it is possible to synthesize two agents that look the same, but have different behaviours—we exploit this ability to paint new behaviour onto previously synthesized agents (Figure 10b).

### 4.2. Synthesizing agents

We paint with elements and we simulate with agents. When we switch between painting and simulating, we map elements to agents, and vice versa (Figure 5). An element in *LifeBrush* contains an additional data area. To map an agent to an element, we copy the agent (with its components) into the additional data area. Then, we set the appearance identifier and the behaviour identifier of the element's attribute vector. Now represented as elements, the element synthesis framework can use the neighbourhood similarity function (Equation 1) to generate new elements along the brush-path. Figure 6 contains an example mapping for our ATP Synthase agent.

Our element-synthesis framework (see Davison et al. [9]) is a separate plugin to *LifeBrush*. Its representation of an element is

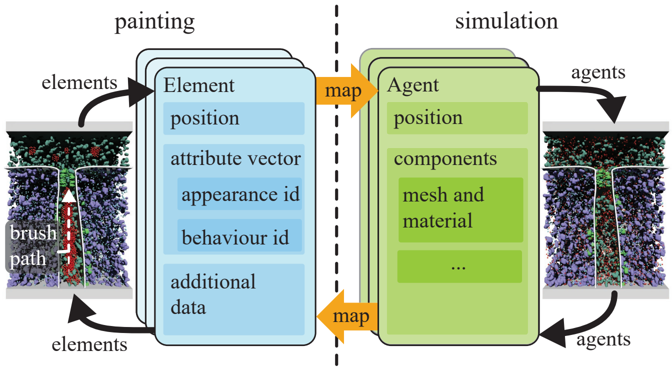



Fig. 5: Our generative tools synthesize elements. When we simulate we map elements into agents. When we paint again, we map the agents back into elements. The additional data area of the element stores the agent state and configuration. The position maps to the element position. Each unique combination of agent appearance and behaviour gets a unique integer identifier. We store the appearance identifier and the behaviour identifier in the element attribute vector, which we use to compare the similarity of two elements during element synthesis.

compact, efficient and separate from an agent. In future work, we would like to include other element synthesis algorithms with their internal representations. Mapping allows us, and possibly other users of *LifeBrush*, to keep the representation of elements separate from our representation of an agent.

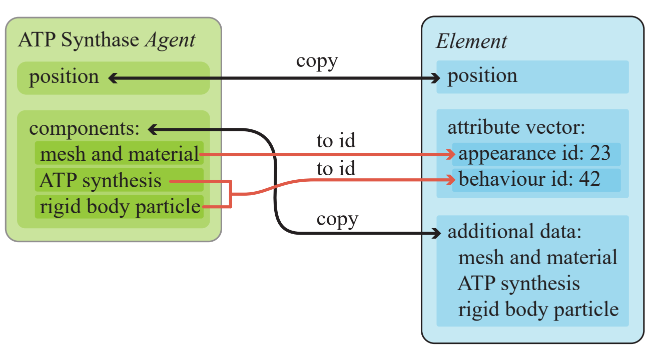



Fig. 6: Mapping an ATP Synthase Agent to an Element. An ATP Synthase Agent has a component for ATP synthesis behaviour, a component for rigid-body interaction through the Nvidia Flex particle physics engine [58, 59], a mesh component with associated material, and a static position component to keep it anchored to the membrane surface. The unique combination of these component classes gets an identifier (for example, 42), which we map to the behaviour identifier of the Element. The mesh and material properties get another unique identifier (for example, 23) that maps to the appearance component of the Element. We copy components to the Element additional data area. Mapping the Element to an Agent copies the additional data area component back to the Agent.

## 5. Sketch-based simulation design in virtual reality

Inspired by physical pencil-and-paper interactions, sketch-based interfaces are used extensively for 3D modelling [45]. A challenge with 2D sketch-based interfaces is how to embed what is fundamentally a 2D curve created by a 2D input device (mouse and keyboard or a digital pen) into a 3D environment. Recent commodity hand-held VR controllers are 3D input devices, tracking position and orientation, that let the user sketch curves directly in 3D space (see Google's Tilt Brush [53]).

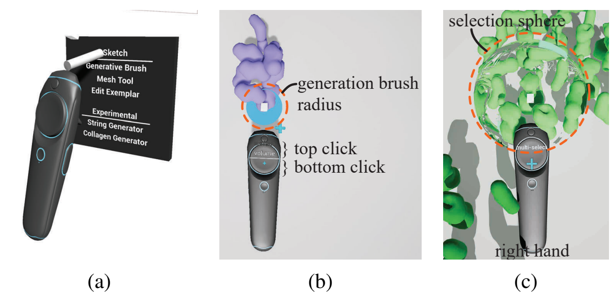



Fig. 7: VR controller operation. (a) The user switches between different tools and settings by pointing their controller at a menu in VR. The size of the generation brush (b) and selection brush (c) is controlled by the analogue trigger button on the VR controller. (b) Clicking the top or bottom of the trackpad toggles different tool modes.

In *LifeBrush*, VR hand-held controllers are used for sketch-based interactions, navigation gestures and for interacting with a VR menu system (Figure 7). Through VR controllers the user interacts with the VR menu, to switch between different tools and to enter or leave the simulation mode.

We support room-scale VR navigation (Figure 2) and navigation gestures. Like an astronaut pulling his/her way through a space station, the **grab** gesture can be used to pull oneself through the world.

The **generative-brush** path $B$ is composed of a set of spheres which have a position and a radius $(bp_i, br_i)$ (Figure 9a). As the user sketches with the brush, the VR controller's analogue trigger button is used to set the radius of the brush spheres. The generative-brush synthesizes new elements within the set of brush spheres $B$. However, when it passes over previously synthesized elements, the position of those elements and the attribute vector are updated to reflect the example palette selection. Elements outside of the brush path are not affected. There are useful applications for this; for example, we use the generative brush to add ATP synthase behaviour to agents in a scene that did not have this behaviour before (Figure 10c).

With the **filler tool,** (Figure 9b) the user identifies a fill point where there are no elements, then we synthesize elements from that point until there is no more room to do so. The **eraser** (Figure 9c) removes elements within a certain distance along a brush path. The **selection brush** selects agents in a radius around the brush.

### *5.1. Assembling agents and desinging examples in the palette*

The example palette is a space where the user designs arrangements of agents and configures their behaviour and other properties (Figure 2). To sketch agents into the simulation, we select agents from the palette, and an example-based synthesis algorithm uses the example to create agents along the brush path. See Davison et al. [9] for a more detailed description of this algorithm.

To create new example arrangements, or to modify existing arrangements, the user grabs agents with a VR controller to move the agents around. We similarly duplicate agents. To create a new agent, the user drags a mesh from the mesh library into the palette (Figure 8). The meshes are created with our sculpting tool (Section 6) or in an external 3D modelling program like Maya [60]. We configure the newly created agent with a property editor interface in VR. To add behaviours to the agent, the user selects the behaviour from a behaviour component class library using a drop-down list. Some parameters, such as numbers, can be difficult to modify with the VR interface, in which case the user can fall back to Unreal Engine's 2D mouse and keyboard interface. To move an agent, the user grabs it with a VR controller (by pulling the analogue trigger), to resize they pull it apart with two hands. Another button allows the user to duplicate an agent.

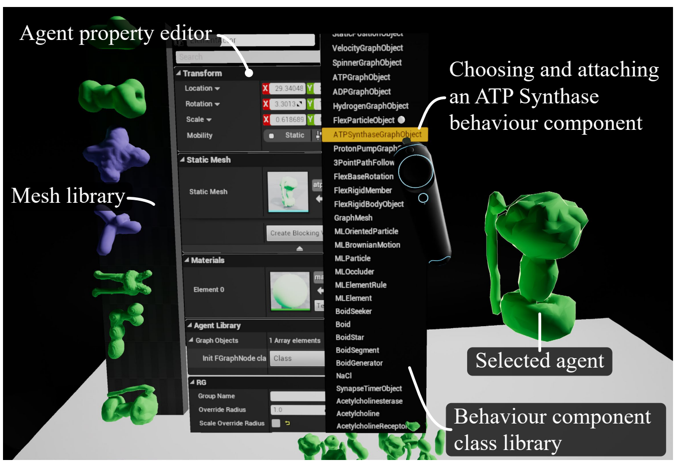



Fig. 8: Editing agent behaviour. In this screenshot, we just created an ATP synthase agent by dragging it from the mesh library to the right. We select an ATP Synthase behaviour from a library of behaviour component classes to give it that behaviour. We configure the properties of the agent, including the newly added ATP synthase component in a property editor.

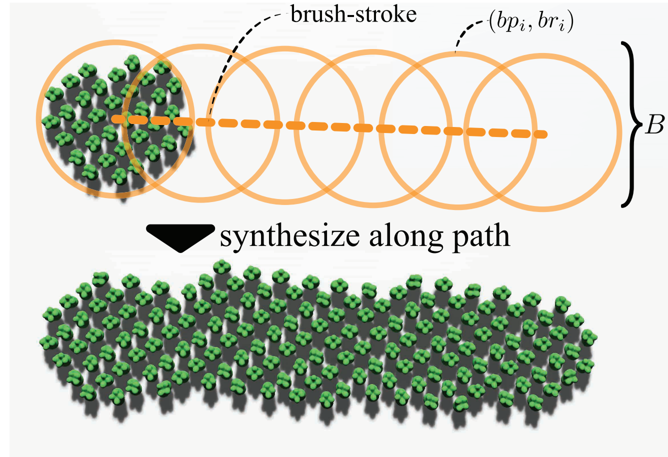
### 5.2. *An example LifeBrush session*

In Figure 10 we describe an example iterative design session using *LifeBrush*. The session illustrates how a user can use *LifeBrush* to experiment with a simulation, using our sketch-based painting tools.

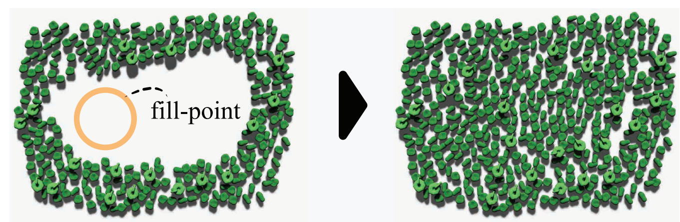
## 6. Implicit surface modeling in VR

The sculpting tool allows the user to create the 3D geometry for molecular agents (Figure 11) and the simulation geometry (Figure 13). As the user paints, the sculpting tool modifies a 3D scalar field of density values, from which a surface reconstruction algorithm (Lorensen and Cline [10]) converts the scalar field into a 3D mesh. The user controls the size of the sculpting brush with an analogue trigger button. Thus, it is possible to create both fine and coarse meshes with the tool.

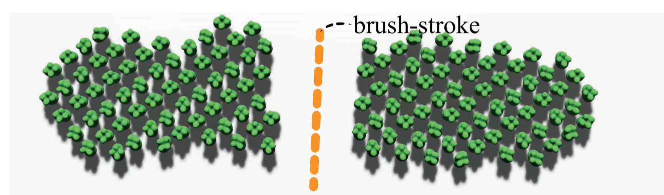
The scalar field is arbitrarily large. To efficiently construct a 3D mesh from the field in real-time we break the field down into chunks (Figure 12). Then, for each recently modified chunk, we efficiently construct a small mesh using marching cubes [10]. The field is broken down into a sparse collection of chunks. If there are no non-zero values in a chunk, it does not consume memory. A chunk is a small grid of density values—for example, a $32^3$ grid of scalar values. The chunks are so small,




(a) **Generative-Brush** New molecular-agents are synthesize along the brush path within the set of brush points $B$.




(b) **Filler Tool** We fill the empty region from the fill-point until there is no more space to add new molecular-agents.

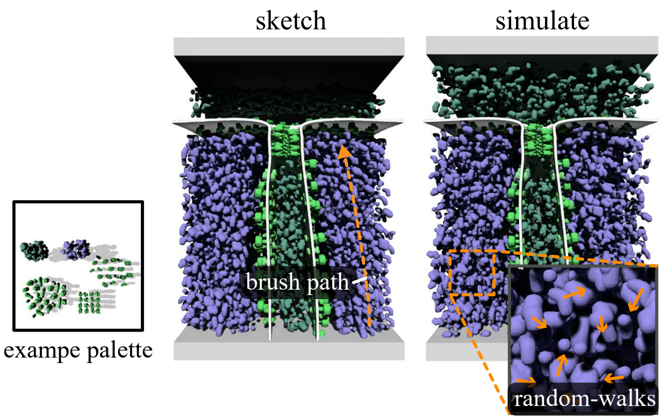



(c) **Eraser** Removing elements along the brush stroke (orange dashed-line).

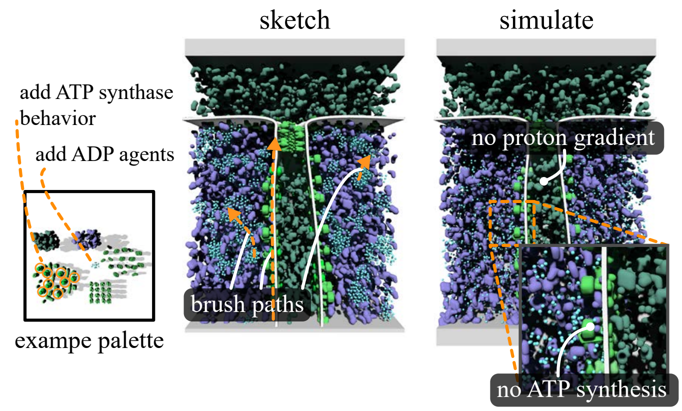
Fig. 9: Our sketch-based tools applied to a planar surface.

that we can construct the chunk meshes in real-time on a 5960x Intel processor running at 3.0 GHz. Before the user paints elements on the mesh, we merge the small chunk meshes into one mesh. We also have an option to trim the mesh to the simulation bounds (Figure 13).

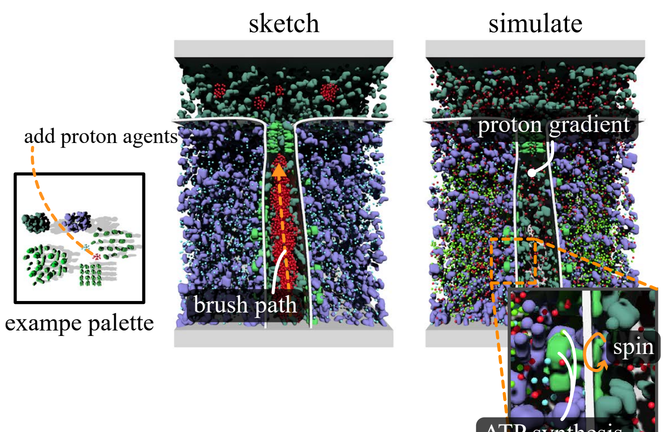
The sculpting tool supports two modes, addition or subtraction. We increment the scalar value of cells overlapping the brush point over time. Cells closer to the brush point increase in value faster. Let $s_i$ be the scalar value at position $p_i \in \mathcal{R}^3$ and let $p_t$ be the position of the sculpting tool point and $r_t$ be the current radius of the tool controlled by the trigger. At each tick of the Unreal Engine [56] the updated value of the scalar field at $p_i$ is $s_i' = s_i + \delta t * (1 - |p_t - p_i|_2 / r_t)$. If $p_i$ is further than $r_t$ from $p_t$ we do not update the value of the scalar field $s_i$. This tool is good for creating smooth rounded objects. With two controllers, it is possible to use this same technique to paint with capsules instead of spheres. In this mode, instead of measuring the distance of a cell from the brush point, we measure the distance of the cell from the line segment between the two controllers and update the scalar field accordingly. In the future,



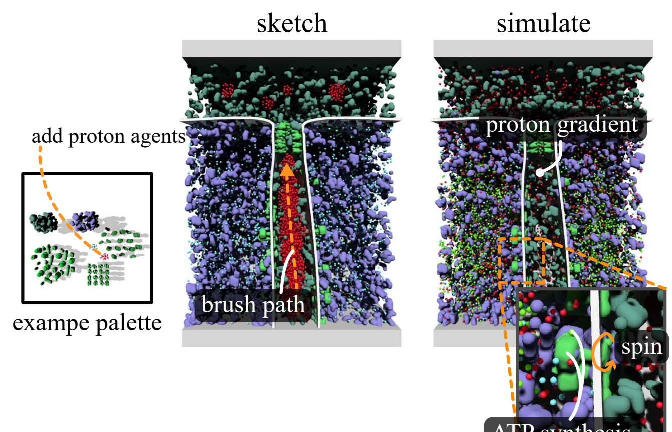

(a) Here we are sketching the initial state of a mitochondrion simulation. We painted the agents in this simulation from the example palette. At this point, the agents only have a random-walk behaviour and the simulation models an inactive mitochondrion.




(b) ATP synthase combines phosphate molecules with adenosine diphosphate (ADP) to create adenosine triphosphate molecules (ATP). We add this behaviour to the ATP synthase agents in the palette (using the editor in Figure 8). We brush over the old ATP synthase agents in the simulation to add the new ATP synthase behaviour from the example. We also paint ADP into the simulation. When we simulate, ADP binds to ATP synthase. However, we need a proton gradient to drive ATP synthase to produce ATP.




(c) Here we add proton agents (that represent a large number of protons) to the example palette and paint protons into the cristae (the central region). When there are more protons on the cristae side of the membrane relative to the other, this creates a charge gradient that drives protons through ATP synthase, causing it to spin and produce ATP.




(d) Eventually, the proton gradient equalizes, and ATP synthase stops producing ATP. In this step, we add a proton pump behaviour to some of the agents in the example palette and paint that new behaviour onto the proton pumps in our simulation. We simulate and observe the restoration of the proton gradient. Eventually, ATP synthase starts spinning again and produces ATP.

Fig. 10: A design session where we used *LifeBrush* to paint and explore a mitochondrion simulation (video: `https://youtu.be/HYLvN2qijeA`).

we would like to add other tools for flattening the implicit surfaces and creating sharp geometries. We chose marching cubes [10] due to the simplicity of the implementation, an alternative and more sophisticated technique is dual-contouring [61].

We use the sculpting tool to create geometry in the environment, hence the user does not have to break immersion to use a third party modelling tool like Maya [60] or Blender [62]. It is also possible to use geometry imported from 3D modelling tools at the same time. In Figure 13 we use the sculpting tool to increase the internal surface area of our mitochondrion and hence the rate of ATP synthesis.

We integrated the sculpting tool with our discrete element texture synthesis system [9]. We identify the disconnected islands in the sculpted mesh and break the mesh into sections. In our discrete element framework, we calculate an orientation field for each section using Crane et al.'s [63] fast algorithm.

The agent-building tool (Section 5.1) contains a library of previously sculpted meshes and imported 3D meshes from external 3D modelling programs. We generated some of the agent-meshes from x-ray crystallography data available from the protein data bank [64]. Creating meshes from the protein data bank takes time, in tools like Maya. Furthermore, the protein databank is incomplete. Therefore, we found the sculpting tool useful for quickly sketching protein meshes, which we may later replace with true-to-life versions. Figure 14 contains some example proteins that we sketched based on 2D illustrations by David Goodsell [2].

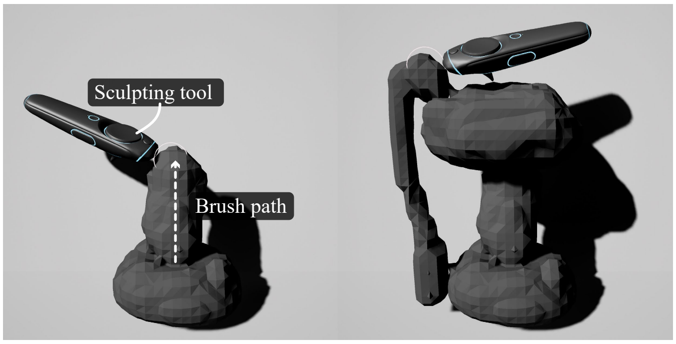
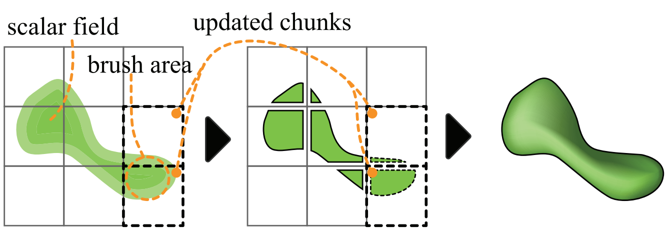
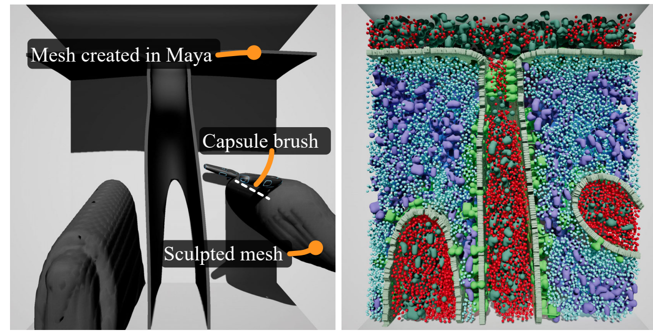



Fig. 11: Sketching the surface of an ATP synthase molecule using our sketch-based sculpting tool in VR. (left) New material is added along the brush path. (right) The complete ATP synthase mesh.



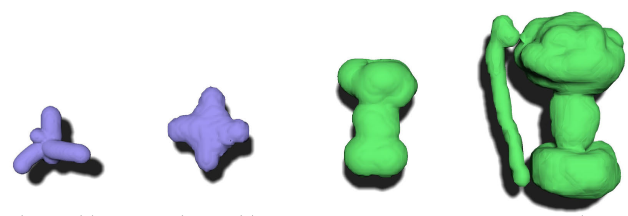

(a) Scalar field (b) Chunked mesh (c) Merged mesh

Fig. 12: (a) The scalar field is broken down into small chunks, typically with dimension $32^3$. When the brush modifies any value in a chunk, we mark the chunk as dirty. (b) We construct a new mesh for each dirty chunk using marching-cubes [10]. (c) Finally, we merge the meshes when the user stops using the sculpting tool.




(a) (b)

Fig. 13: Sculpting the mitochondrial environment. (a) We use the capsule brush to sculpt new cristae regions between the two controllers. The scene also contains a mesh that we designed in Maya. (b) We trim the sculpted meshes to the simulation bounds and paint lipids and molecular-agents into the scene. The mitochondrion now has more internal surface area for ATP synthesis than in Figure 2c.



Cristae widget Cristae widget Proton pump ATP Synthase

Fig. 14: Sculpted molecular agents. These are some molecules that we created with our sculpting tool. The purple proteins are widgets whose function is to fill the visual space of the mitochondrion (we only simulate their rigid body interaction with other molecules). The green proteins are a proton pump and an ATP synthase.

## 7. Iteractive visualization with simulation timelines

In the original *LifeBrush* system it was challenging to observe interactions between molecular-agents in the dense and busy mitochondrion environment [7]. Our proposed solution to this problem is a history based visualization that highlights molecular-agent interactions.

The timeline data structure is a historical record of interaction events and the state of the simulation at different points in time. Whenever an agent triggers an interaction event, such as an ATP-Synthase molecule creating an ATP molecule, we record the event in the timeline. The event stores the time of the interaction, the agent that triggered the interaction and the other interacting agents. The timeline also records the position of the agents every $k$ seconds (the default value for $k$ is 0.25s). We have developed two interactive visualizations. *Agent pathlines* show the path agents have taken through the simulation. An *event trace* visualizes a sequence of interactions between a set of agents.

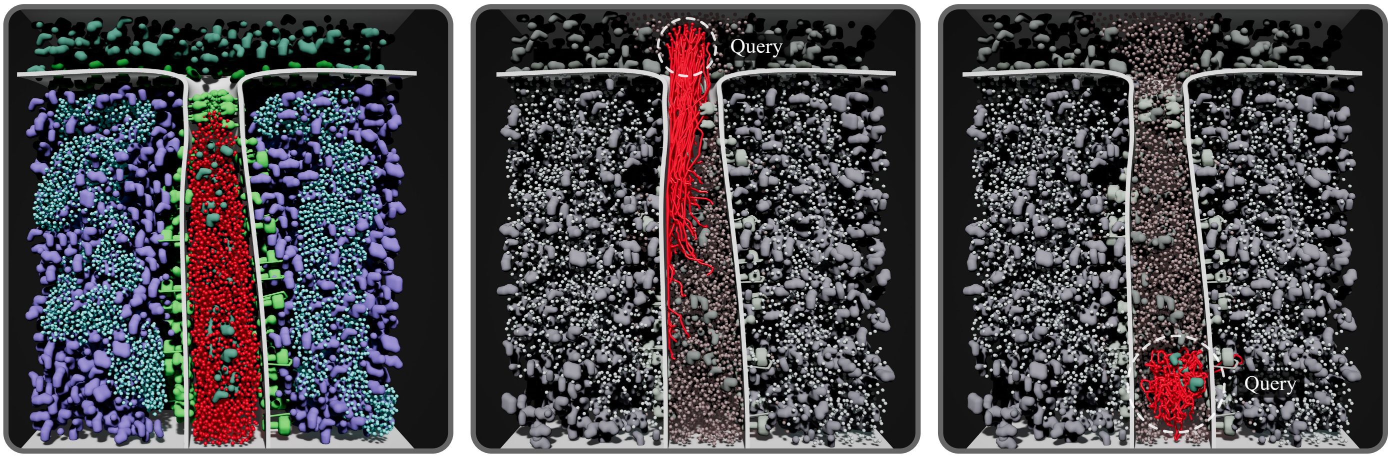
### 7.1. Agent pathlines

Pathlines in scientific visualization have been used to trace the flow of virtual particles seeded from a starting position through an unsteady vector field [65]. We visualize the paths of molecular-agents as they course their way through the simulation (Figure 15). An underlying vector field does not drive the agent trajectory. Instead, the trajectory of the agents is implicit to the interactions between the agents.

*Agent pathlines* twist and weave their way through the simulation space where other molecular-agents occlude them from view. To solve this problem, we hide occluding molecular-agents that lie between the user's eyes and the pathlines. We do this efficiently in real-time by raycasting, from the eye position in the direction of each agent in the scene, against a bounding-volume-hierarchy of the pathline visualization [66]. We desaturate the appearance of the molecular-agents that were not queried by the user, to make the pathlines easier to see in VR.

The user interacts with the pathline visualization by querying a set of molecular-agents with a selection brush. In Figure 15 we query the simulation after it has run for a few seconds. The pathline visualization reveals some interesting observations: 1) the central region was under pressure, that pushed the hydrogen agents (traces in red) to a region of lower pressure (the top in Figure 15b), and 2) there was a bias in the physics simulation that pushed agents from the left towards the top, instead of uniformly from either side of the central region. We are unsure where this bias comes from, but perhaps it is a property of the physics library we are using [58].

We found that the visualization tool had unexpected utility for debugging. For example, we did not notice when we were building our simulations, that some molecular-agents were tunnelling through the collision geometry, into regions they did




(a) (b) (c)

Fig. 15: Pathlines reveal the course taken by molecular agents in a simulation. (a) The initial state of a mitochondrial simulation, with hydrogen agents concentrated in the central region. After a few seconds we paused the simulation. (b) Querying the agents at the top (white dotted line), we see that the hydrogen agents moved from the central region to the top. There is a notable bias of left originating hydrogen agents. (c) With nowhere to go, the molecular-agents at the bottom followed a wandering path.

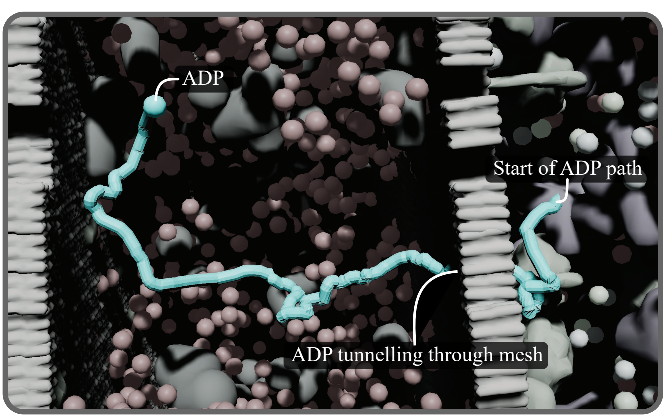



Fig. 16: Debugging with pathlines. We did not configure ADP to tunnel through the collision geometry like it is in this pathline visualization. We fixed the issue by modifying the collision parameters for the mesh.

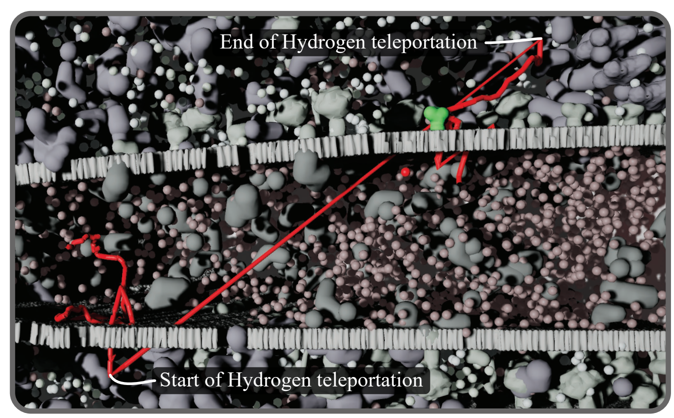



Fig. 17: Debugging with event traces. While visualizing proton pump behaviour, we noticed that hydrogen was teleporting across the simulation at random intervals. This visualization helped us notice and track down the bug in our code.

not belong (Figure 16). We corrected the problem by modifying particle-surface collision properties. In another simulation, we observed molecular-agents teleporting across the simulation (Figure 17). The teleportation was due to a bug in the ATP synthase agent behaviour.

### 7.2. Interaction event traces

When an event occurs, we place a 3D glyph (a coloured cube) at the location of the event. The user interacts with the *event trace* visualization by selecting event glyphs with a VR controller.

For each selected event glyph, we trace forwards and backwards in the timeline from when the event occurred, looking for other events that reference the agents affected by the event. We do this recursively, up to a certain depth, and collate those events. Then we visualize the location of each collated event and produce a pathline visualization for the involved agents.

In Figure 18, we query an ATP synthesis event. The *event trace* visualization tells a story, where a hydrogen agent (red pathline) was driven through the ATP synthase molecule, giving it the energy to convert an ADP molecule into ATP, later that hydrogen agent was pumped back into the cristae (the central region of Figure 18a). Querying multiple events reveals the network of molecular-agent interactions in our mitochondrion (Figure 18c).

*Event traces* take a 4D dataset (spatiotemporal positions) and present it as a pathline. With the simulation paused, the user can take their time and explore a series of molecular interactions. As we mentioned, *event traces* tell the store of ATP synthesis and we think the visualization could have applications for explaining other molecular processes in mesoscale environments.

3D animations like Harvard's Biovisions project [3], use scripted and carefully animated video sequences to tell a story. Telling such a story with a simulation can be challenging, be-

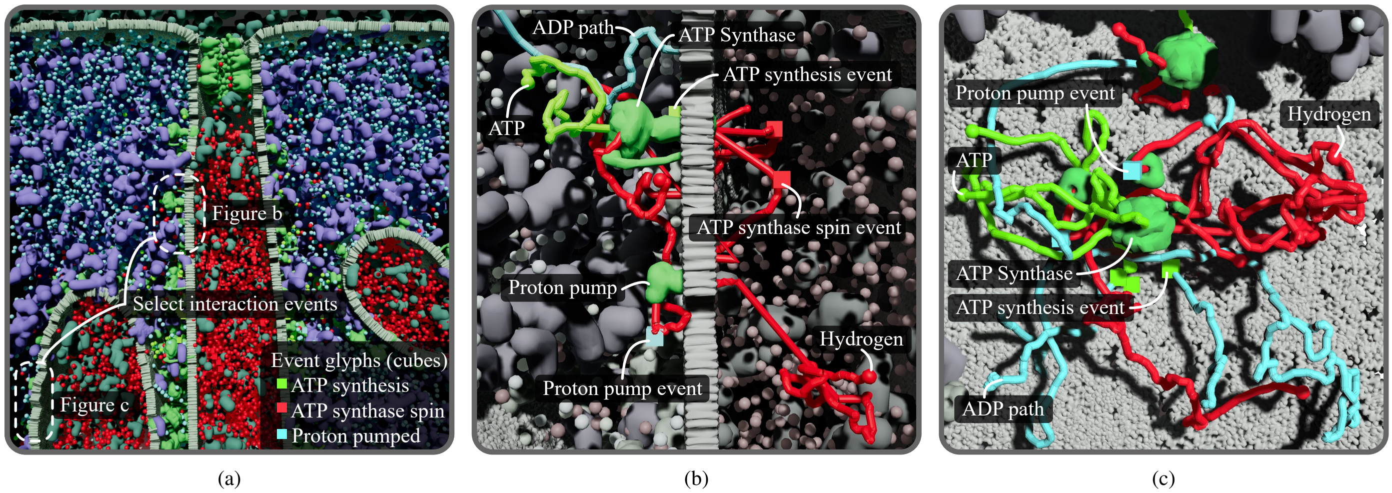



Fig. 18: *Event trace* visualizations reveal sequences of molecular-agent interactions. (a) The user selects event glyphs (the coloured cubes embedded in the simulation) with a brush tool, in the regions indicated by the white dashed lines. (b) Zooming into the central region, a molecular story unfolds. Hydrogen, pushed by a proton gradient, is driven through ATP synthase, giving it mechanical energy to convert ADP into ATP, which moves away. Meanwhile, the hydrogen agent wanders into a proton pump and is pushed back into the central region to maintain a proton gradient. (c) Our event trace visualization reveals the network of molecular-agent interactions over our mitochondrial membrane.

cause we may have to wait a long time for just the right sequence of interactions, that fit the story, to occur. Le Muzic et al. [67] reduce the wait between events with a type of random interpolation. However, with pathlines, we can visualize the time between events, while also visualizing past and subsequent interactions. Without editing the screenshots in an image editor (other than to add annotations), we were also able to use our visualization to explain mitochondrial function in several 2D figures in this article (such as Figure 1).

## 8. Discussion and future work

In our original *LifeBrush* paper, we painted and brought to life a mesoscale illustration of the mitochondrion, in a VR Cyberworld [7]. In this extended article, we describe additional tools for sculpting environmental and molecular-agent geometry with VR. Our example mitochondrion simulation is dense and chaotic, replicating a real biomolecular system. Therefore we have also extended our system with *pathline* and *event trace* visualizations to understand that chaotic space.

Event trace visualizations are a useful tool for explaining and exploring agent interactions. We also found them useful for debugging. In the future, we would like to add a replay option. With a replay option, a *LifeBrush* simulation could be used in educational settings for students to interactively explore and experiment with mesoscale systems.

To generate and simulate the results in this article, we used an Intel 5960x processor with eight cores running at 3.0 GHz (the processor was released in 2014), 16 GB of RAM and an Nvidia GTX 1080 GPU. We used an HTC Vive VR headset and controllers. Our results run at 90 frames-per-second in VR with about 10,000 agents. In the future, we could improve performance by multithreading our simulations and using level-of-detail techniques to reduce GPU overhead.

The target users for *LifeBrush*, include scientific illustrators, modellers, and computer scientists building mesoscale illustrations. We demonstrated an interactive sketch-based system; however, creating even simple molecular-agents requires writing C++ code. For users who can program, this is still a significant and time-consuming limitation. In the future, we plan to explore interactive and visual ways of defining molecular agent interactions to augment the programming interface. One possible solution is to provide the system with example agent interactions and then derive automatic rules for those interactions, such as with prototypical situation-action-pairs [68].

Our sculpting tool is limited to capsular or spherical brushes. We want to add more brush types and functions. Another possibility is to add subdivision modelling tools.

In the future, we would also like to improve the accuracy of our mitochondrion illustration, especially with more function and behaviour. It would also be necessary to validate it against accepted models as well as biological in vivo experiments. However, even with our naive implementation, we think that it still has illustrative value. Future work will explore creating other mesoscale illustrations and simulations with our system.

In theory, we can extend our agent-to-element mapping to other agent-based systems and discrete element texture synthesis algorithms. For example, *LifeBrush* could be useful for painting crowd and ecosystem simulations. We would also like to incorporate Roveri et al.'s [44] algorithm for repetitive structure synthesis, among others.

Previous systems use recipe files to automatically pack molecules into mesoscale environments [13, 14, 6]. In contrast to these systems, we let the user interactively paint molecular agents into the mesoscale environment. We also demonstrated immediately bringing that environment to life within the same session. An exciting interaction is using our sketch-based tools to experiment with the simulation (Section 5.2) interactively.

Finally, an exciting possibility is collaborative editing and playback with *LifeBrush* in a multiplayer environment and across non-VR devices (like tablets and desktop computers). We imagine applications of our method for interactive illustration and teaching of mesoscale environments.

## References


[1] Zick, M, Rabl, R, Reichert, AS. Cristae formationlinking ultrastructure and function of mitochondria. Biochimica et Biophysica Acta (BBA)-Molecular Cell Research 2009;1793(1):5–19.

[2] Goodsell, DS. The machinery of life. Springer Science & Business Media; 2009.

[3] Harvard BioVisions, . The inner life of the cell (video). 2007. URL: `http://biovisions.mcb.harvard.edu`.

[4] Esmaeili, A, Davison, T, Wu, A, Alcantara, J, Jacob, C. Prokaryo: an illustrative and interactive computational model of the lactose operon in the bacterium escherichia coli. BMC bioinformatics 2015;16(1):311.

[5] Yuen, D, Jacob, C. Eukaryo: An agent-based, interactive simulation of a eukaryotic cell. In: Artificial Life Conference 2016. 2016, p. 562.

[6] Klein, T, Autin, L, Kozlíková, B, Goodsell, DS, Olson, A, Gröller, ME, et al. Instant construction and visualization of crowded biological environments. IEEE transactions on visualization and computer graphics 2018;24(1):862–872.

[7] Davison, T, Samavati, F, Jacob, C. Lifebrush: Painting interactive agent-based simulations. In: 2018 International Conference on Cyberworlds (CW). 2018,.

[8] Reynolds, CW. Flocks, herds and schools: A distributed behavioral model. In: ACM SIGGRAPH computer graphics; vol. 21. ACM; 1987, p. 25–34.

[9] Davison, T, Samavati, F, Jacob, C. Interactive example-palettes for discrete element texture synthesis. Computers & Graphics 2018;URL: `http://www.sciencedirect.com/science/article/pii/S0097849318301778`. doi:`https://doi.org/10.1016/j.cag.2018.10.016`.

[10] Lorensen, WE, Cline, HE. Marching cubes: A high resolution 3d surface construction algorithm. SIGGRAPH Comput Graph 1987;21(4):163–169.

[11] Kozlikova, B, Krone, M, Lindow, N, Falk, M, Baaden, M, Baum, D, et al. Visualization of biomolecular structures: state of the art. In: Eurographics Conference on Visualization (EuroVis)-STARs. The Eurographics Association; 2015, p. 061–081.

[12] Miao, H, Klein, T, KouÅil, D, Mindek, P, Schatz, K, Grller, ME, et al. Multiscale molecular visualization. Journal of Molecular Biology 2018;URL: `http://www.sciencedirect.com/science/article/pii/S0022283618310490`. doi:`https://doi.org/10.1016/j.jmb.2018.09.004`.

[13] Martínez, L, Andrade, R, Birgin, EG, Martínez, JM. Packmol: a package for building initial configurations for molecular dynamics simulations. Journal of computational chemistry 2009;30(13):2157–2164.

[14] Johnson, GT, Autin, L, Al-Alusi, M, Goodsell, DS, Sanner, MF, Olson, AJ. cellpack: a virtual mesoscope to model and visualize structural systems biology. Nature methods 2015;12(1):85.

[15] Koch, TB, Kouřil, D, Klein, T, Mindek, P, Viola, I. Semantic screen-space occlusion for multiscale molecular visualization. In: Proceedings of the Eurographics Workshop on Visual Computing for Biology and Medicine. Eurographics Association; 2018, p. 197–201.

[16] Kouřil, D, Čmolík, L, Kozlikova, B, Wu, HY, Johnson, G, Goodsell, DS, et al. Labels on levels: labeling of multi-scale multi-instance and crowded 3d biological environments. IEEE transactions on visualization and computer graphics 2019;25(1):977–986.

[17] Muzic, ML, Autin, L, Parulek, J, Viola, I. cellVIEW: a Tool for Illustrative and Multi-Scale Rendering of Large Biomolecular Datasets. In: Bhler, K, Linsen, L, John, NW, editors. Eurographics Workshop on Visual Computing for Biology and Medicine. The Eurographics Association. ISBN 978-3-905674-82-8; 2015,doi:`10.2312/vcbm.20151209`.

[18] Haefner, JW. Modeling biological systems: principles and applications. Springer Science & Business Media; 2012.

[19] Jacob, C, Burleigh, I. Biomolecular swarmsan agent-based model of the lactose operon. Natural Computing: an international journal 2004;3(4):361–376.

[20] Jacob, C, Barbasiewicz, A, Tsui, G. Swarms and genes: Exploring $\lambda$-switch gene regulation through swarm intelligence. In: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on. IEEE; 2006, p. 2535–2542.

[21] Sarpe, V, Jacob, C. Simulating the decentralized processes of the human immune system in a virtual anatomy model. In: BMC bioinformatics; vol. 14. BioMed Central; 2013, p. S2.

[22] Karr, JR, Sanghvi, JC, Macklin, DN, Gutschow, MV, Jacobs, JM, Bolival Jr, B, et al. A whole-cell computational model predicts phenotype from genotype. Cell 2012;150(2):389–401.

[23] Jacob, C, von Mammen, S, Davison, T, Sarraf-Shirazi, A, Sarpe, V, Esmaeili, A, et al. Lindsay virtual human: Multi-scale, agent-based, and interactive. In: Advances in Intelligent Modelling and Simulation. Springer; 2012, p. 327–349.

[24] Wu, A, Davison, T, Jacob, C. A 3d multiscale model of chemotaxis in bacteria. In: Artificial Life Conference 2016. 2016, p. 546.

[25] Shirazi, AS, Davison, T, von Mammen, S, Denzinger, J, Jacob, C. Adaptive agent abstractions to speed up spatial agent-based simulations. Simulation Modelling Practice and Theory 2014;40:144–160.

[26] Boyd, JE, Hushlak, G, Jacob, CJ. Swarmart: interactive art from swarm intelligence. In: Proceedings of the 12th annual ACM international conference on Multimedia. ACM; 2004, p. 628–635.

[27] Miner, D, Kasch, N. Swarmvis: a tool for visualizing swarm systems. UMBC Computer Science 2008;636.

[28] Wei, LY, Lefebvre, S, Kwatra, V, Turk, G. State of the art in example-based texture synthesis. In: Eurographics 2009, State of the Art Report, EG-STAR. Eurographics Association; 2009, p. 93–117.

[29] Schachter, B, Ahuja, N. Random pattern generation processes. Computer Graphics and Image Processing 1979;10(2):95–114.

[30] Efros, AA, Leung, TK. Texture synthesis by non-parametric sampling. In: Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on; vol. 2. IEEE; 1999, p. 1033–1038.

[31] Wei, LY, Levoy, M. Texture synthesis over arbitrary manifold surfaces. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. ACM; 2001, p. 355–360.

[32] Kwatra, V, Essa, I, Bobick, A, Kwatra, N. Texture optimization for example-based synthesis. In: ACM Transactions on Graphics (TOG); vol. 24. ACM; 2005, p. 795–802.

[33] Cohen, MF, Shade, J, Hiller, S, Deussen, O. Wang tiles for image and texture generation; vol. 22. ACM; 2003.

[34] Li, C, Wand, M. Precomputed real-time texture synthesis with markovian generative adversarial networks. In: European Conference on Computer Vision. Springer; 2016, p. 702–716.

[35] Vanhoey, K, Sauvage, B, Larue, F, Dischler, JM. On-the-fly multi-scale infinite texturing from example. ACM Transactions on Graphics (TOG) 2013;32(6):208.

[36] Han, C, Risser, E, Ramamoorthi, R, Grinspun, E. Multiscale texture synthesis. In: ACM Transactions on Graphics (TOG); vol. 27. ACM; 2008, p. 51.

[37] Glanville, S. Texture bombing. GPU Gems: Programming Techniques, Tips, and Tricks for 2004;.

[38] Wang, L, Shi, Y, Chen, Y, Popescu, V. Just-in-time texture synthesis. In: Computer Graphics Forum; vol. 32. Wiley Online Library; 2013, p. 126–138.

[39] Lefebvre, S, Hornus, S, Neyret, F. Texture sprites: Texture elements splatted on surfaces. In: Proceedings of the 2005 symposium on Interactive 3D graphics and games. ACM; 2005, p. 163–170.

[40] Hurtut, T, Landes, PE, Thollot, J, Gousseau, Y, Drouillhet, R, Coeurjolly, JF. Appearance-guided synthesis of element arrangements by example. In: Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering. ACM; 2009, p. 51–60.

[41] Landes, PE, Galerne, B, Hurtut, T. A shape-aware model for discrete texture synthesis. In: Computer Graphics Forum; vol. 32. Wiley Online Library; 2013, p. 67–76.

[42] Ma, C, Wei, LY, Tong, X. Discrete element textures. In: ACM Transactions on Graphics (TOG); vol. 30. ACM; 2011, p. 62.

[43] Ijiri, T, Mech, R, Igarashi, T, Miller, G. An example-based procedural system for element arrangement. In: Computer Graphics Forum; vol. 27. Wiley Online Library; 2008, p. 429–436.

[44] Roveri, R, Öztireli, AC, Martin, S, Solenthaler, B, Gross, M. Example based repetitive structure synthesis. In: Computer Graphics Forum; vol. 34. Wiley Online Library; 2015, p. 39–52.

[45] Olsen, L, Samavati, FF, Sousa, MC, Jorge, JA. Sketch-based modeling: A survey. Computers & Graphics 2009;33(1):85–103.

[46] Deussen, O, Hanrahan, P, Lintermann, B, Měch, R, Pharr, M, Prusinkiewicz, P. Realistic modeling and rendering of plant ecosystems. In: Proceedings of the 25th annual conference on Computer graphics and interactive techniques. ACM; 1998, p. 275–286.

[47] Emilien, A, Vimont, U, Cani, MP, Poulin, P, Benes, B. Worldbrush: Interactive example-based synthesis of procedural virtual worlds. ACM Trans Graph 2015;34(4):106:1–106:11.

[48] Gain, J, Long, H, Cordonnier, G, Cani, MP. Ecobrush: Interactive control of visually consistent large-scale ecosystems. In: Computer Graphics Forum; vol. 36. Wiley Online Library; 2017, p. 63–73.

[49] Ketabchi, K, Runions, A, Samavati, FF. 3d maquetter: Sketch-based 3d content modeling for digital earth. In: 2015 International Conference on Cyberworlds (CW). 2015, p. 98–106. doi:10.1109/CW.2015.41.

[50] Samavati, F, Runions, A. Interactive 3d content modeling for digital earth. The Visual Computer 2016;32(10):1293–1309.

[51] Zhu, B, Iwata, M, Haraguchi, R, Ashihara, T, Umetani, N, Igarashi, T, et al. Sketch-based dynamic illustration of fluid systems. In: ACM Transactions on Graphics (TOG); vol. 30. ACM; 2011, p. 134.

[52] Gu, Q, Deng, Z. Formation sketching: an approach to stylize groups in crowd simulation. In: Proceedings of Graphics Interface 2011. Canadian Human-Computer Communications Society; 2011, p. 1–8.

[53] Google LLC, . Tilt brush software. 2016. URL: https://www.tiltbrush.com/.

[54] Afsharchi, M, Far, BH, Denzinger, J. Ontology-guided learning to improve communication between groups of agents. In: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. ACM; 2006, p. 923–930.

[55] Rene, B. Component based object management. Game Programming Gems 2005;5:25–37.

[56] Epic Games, Inc., . Unreal Engine 4 Game Engine. 2019. URL: https://www.unrealengine.com.

[57] Unity Technologies, ApS, . Unity 3d game engine. 2019. URL: https://unity.com/.

[58] Nvidia Corporation, . Flex particle physics library. 2019. URL: https://developer.nvidia.com/flex.

[59] Macklin, M, Müller, M, Chentanez, N, Kim, TY. Unified particle physics for real-time applications. ACM Transactions on Graphics (TOG) 2014;33(4):104.

[60] Autodesk, Inc., . Autodesk maya. 2019. URL: https://www.autodesk.ca.

[61] Ju, T, Losasso, F, Schaefer, S, Warren, J. Dual contouring of hermite data. In: ACM transactions on graphics (TOG); vol. 21. ACM; 2002, p. 339–346.

[62] Blender Foundation, . Blender. 2019. URL: https://www.blender.org/.

[63] Crane, K, Desbrun, M, Schröder, P. Trivial connections on discrete surfaces. In: Computer Graphics Forum; vol. 29. Wiley Online Library; 2010, p. 1525–1533.

[64] Berman, HM, Westbrook, J, Feng, Z, Gilliland, G, Bhat, TN, Weissig, H, et al. The protein data bank. Nucleic Acids Res 2000;28(1):235–242. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC102472/; gkd090[PII].

[65] Turk, G, Banks, D. Image-guided streamline placement. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. ACM; 1996, p. 453–460.

[66] Kay, TL, Kajiya, JT. Ray tracing complex scenes. In: ACM SIGGRAPH computer graphics; vol. 20. ACM; 1986, p. 269–278.

[67] Le Muzic, M, Waldner, M, Parulek, J, Viola, I. Illustrative timelapse: A technique for illustrative visualization of particle-based simulations. In: 2015 IEEE Pacific Visualization Symposium (PacificVis). IEEE; 2015, p. 247–254.

[68] Davison, T, Denzinger, J. The huddle: Combining ai techniques to coordinate a player's game characters. In: Computational Intelligence and Games (CIG), 2012 IEEE Conference on. IEEE; 2012, p. 203–210.