LifeBrush: Painting Interactive Agent-Based Simulations

Conference Paper · October 2018

DOI: 10.1109/CW.2018.00017

Timothy Davison
The University of Calgary
13 PUBLICATIONS **44** CITATIONS

Faramarz F. Samavati
The University of Calgary
139 PUBLICATIONS **1,404** CITATIONS

Preprint to appear at the 2018 International Conference on Cyberworlds

LifeBrush: Painting interactive agent-based simulations

Timothy Davison*, Faramarz Samavati† and Christian Jacob‡
Department of Computer Science
University of Calgary
Calgary, Canada
*Email: *tbdaviso@ucalgary.ca, †samavati@ucalgary.ca, ‡cjacob@ucalgary.ca*

***Abstract*—Building and interacting with 3D agent-based simulations that contain a large number of agents is a significant challenge. What if we want to create an intricate new arrangement of agents, or reconfigure a large number of agents? We present LifeBrush, a cyberworld for interactively painting large and elaborate multi-agent simulations with commodity virtual reality systems that we can then simulate and explore. Our main methodology uses sketch-based discrete element texture synthesis to paint agent arrangments. We define a map to convert agents to elements in this framework when we paint and back to agents when we simulate. Like creating new colors on a paint palette, we create example agent arrangements and configurations in an example palette. We paint new agents into a scene with sketch-based generative brushes. We also use those brushes to reconfigure agents to match examples created in the palette. Then we simulate, pause the simulation and modify the agents with our sketch-based tools. This iteration loop enables new levels of interactivity for the design, simulation, and exploration of agent-based simulations.**

I. Motivation

Agent-based simulations have been used to model biological systems such as swarming insects and birds [1]. Biomolecular processes within cells [2] are another example where agent-based models help to capture complex interactions among many entities. Imagine the intricate arrangement of proteins and their relationships that occur within a small compartment of a biological cell [3]. It is challenging though, to build such simulations. First one has to configure the initial state of the model. Then one has to define the properties and state of the various types of agent interactions that constitute the model dynamics.

Consider zooming inside a biological cell, and navigating to one of its organelles (Figure 1). Let's pick a mitochondrion, the powerhouse of the cell. Here ATP is assembled, a molecule which is used to provide energy to the rest of the cell (for details on mitochondrion function see Section V). Contrary to textbook illustrations, mitochondria are densely packed and highly organized [4].

One method to configure such a molecular simulation is through parameterization [5]. Yuen et al. [6] configured mitochondria in their simulation through a combination of manual and randomized placement of agents. But, would it be possible to create a cyberworld to more intuitively create, bring to life, and interactively control such a simulation?

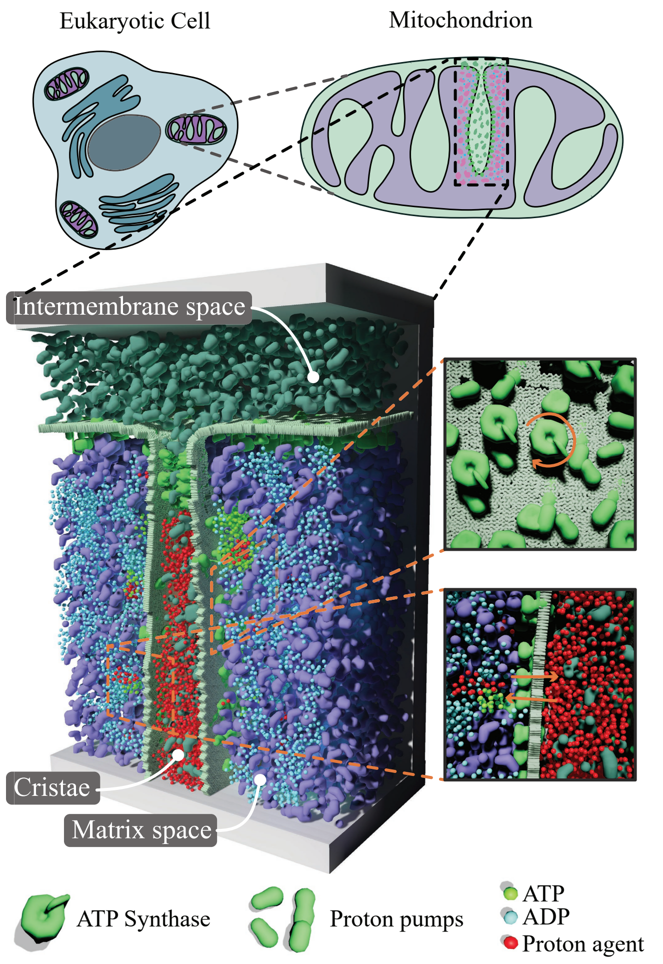


Figure 1. We painted this interactive 3D agent-based mitochondrion simulation with our system. The mitochondrion is an organelle inside of Eukaryotic biological cells. When we press play, the simulation comes to life: spinning ATP synthase converts ADP to ATP, while proton pumps move protons across the inner membrane. We painted this simulation with 20,000 agents in about 5 minutes.

Imagine we "paint" a virtual mitochondrion, then "walk into" its 3D space and "press play" (i.e., bring it to life). Being immersed, we navigate through the simulation, explore and observe. Then, we pause the simulation, adding

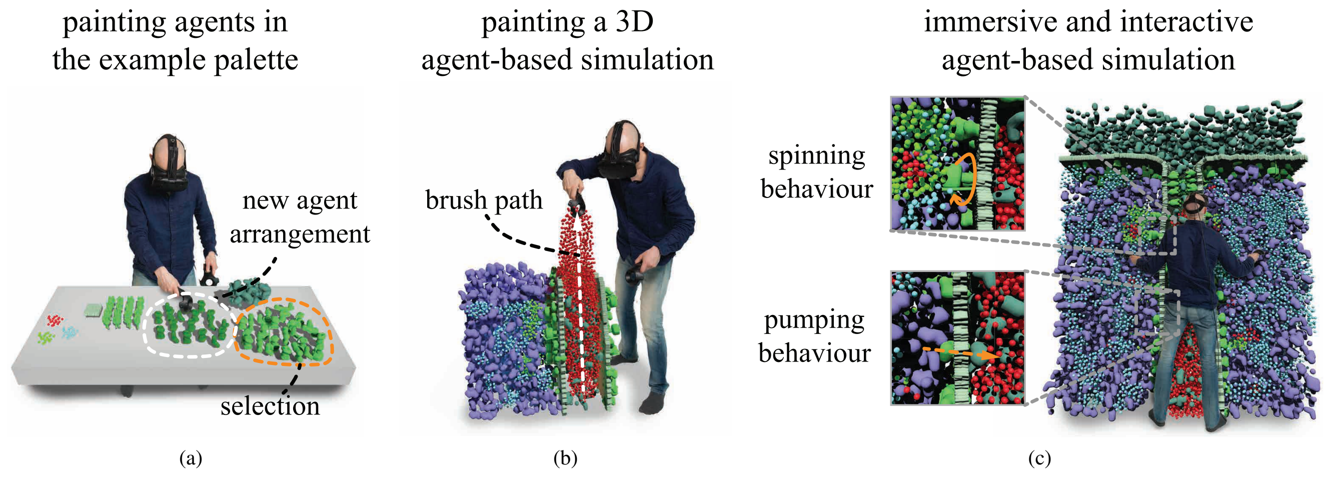


Figure 2. (a) Painting a new arrangement of agents (white outline) into the **example palette** using another arrangement of agents (orange outline). (b) **Painting** the red agents from (a) into an agent-based simulation. (c) Stepping into an **immersive simulation** we can the painted agents come to life. In the close-ups, molecules in the mitochondrion spin and pump red agents along the orange arrow.

new agents. We resume, pause again, and reconfigure the membrane proteins. How can we do this interactively and within a 3D immersive environment (Figure 2 and 3)?

We introduce a novel sketch-based synthesis system to interactively generate, configure and guide the agents in a live 3D simulation. We developed a virtual reality (VR) cyberworld—using commodity VR hardware—where a user can virtually step into an agent-based simulation, such as the mitochondrion. Our sketch-based generative brushes are used to "paint" proteins along a brush-path, in space, or onto 3D surfaces (Section IV).

We use example-based synthesis to create agents arranged and configured based on an example palette. The attributes and spatial arrangement of synthesized agents are determined by the texture synthesis algorithm to match the examples found in the example palette through a combination of generation and optimization steps. Just like an artist would mix colors on a palette and apply them to a canvas, we create example agent palettes from which we paint agent arrangements into the scene (Figure 2a). We support a variety of stochastic and structured distributions. To initially define agents in the palette, we manually place them in the desired arrangement and configure their behaviors and properties. Our generative tools can also be used to sketch new example arrangements derived from previous examples. Our generative framework can both synthesize new agents or reconfigure agents in a simulation to match examples in the palette (Section III).

We base our sketch-based generative tools on example-based discrete element texture synthesis (Section III). In this framework, we represent agents with elements. Elements have a position and attribute vector that compactly and efficiently capture the behavior and properties that define an agent (Section 5). When we paint, we convert the agent example palette and the agent-based simulation to elements, and when we simulate, we convert the elements back into agents. Therefore, our system defines a mapping to convert agents to elements and vice versa. The duality between elements and agents allows us to leverage sophisticated techniques from discrete element texture synthesis [7, 8, 9] to synthesize large and intricate agent-based simulations.

We use a region growing algorithm to synthesize new elements iteratively. This method is fast and suitable for interactive applications [9]. An optimization procedure relaxes newly synthesized elements relative to the example palette. Our system has been carefully designed to be performant for interactive VR applications.

We will also describe how to run these agent simulations in real-time, and interactively pause, sketch and **reconfigure agents** (Section V). Thus, our system, LifeBrush, provides an interactive design and exploration path for complex agent-based simulations.

II. Related work

Here we will use the molecular machinery of life as examples to illustrate our LifeBrush system. Yet, it should be noted that our system can be used for any 3D agent-based simulation.

Molecular Dynamics Construction and Visualization

Harvard BioVisions released a series of carefully animated videos illustrating various molecular processes inside biological cells [10]. UnityMol [11] is a tool for visualizing molecules, and Molecular Rift is VR molecular viewer [12], both of which were designed for visualizing single molecules.

David Goodsell [3] has created highly detailed yet static 2D illustrations of the molecular machinery inside cells.

Many different techniques have been developed for producing 3D visualizations of structures inside of biological cells [13]. With Packmol [14] and CellPack [15] one can randomly pack proteins and molecules onto surfaces and regions inside of a virtual cell according to recipe files. Klein et al. [5] have used GPUs to accelerate the packing process based on parameterized recipes. These methods do not simulate the resulting scenes, though. Our system uses sketch-based synthesis to paint agents into a simulation which can be played interactively. We also present the simulation in VR for increased immersion.

Agent-based modeling

Agent-based approaches have been used to model biological systems like swarming insects and birds [1], without relying on purely mathematical models [16]. Agent-based systems have also been used to model the Lactose operon inside *E. coli* bacteria [17], for gene regulation [18] and immune system models [19]. Along the lines of mathematical whole-cell models [20], agent-based models have been applied to both prokaryotic [2] and eukaryotic cells [6]. Meanwhile, multi-scale agent-based models can simultaneously capture cells and groups of cells at different scales [21, 22]. Automatic abstraction has been used to reduce the computational complexity of such models [23].

Sketch-based synthesis and procedural modeling

Sketch-based interfaces apply the familiarity of real-world tools like pencil and paper to interactive design problems, such as 3D modeling [24]. Ecological simulation has been used to synthesize and render large plant ecosystems [25]. Procedural modeling can generate 3D structures, such as buildings [26]. Meanwhile, statistical sketch-based synthesis has been used to create terrains covered in trees and vegetation [27, 28]. Interactive 3D content modeling has been applied to the digital earth [29, 30]. Example-based discrete element texture synthesis uses a small example of discrete elements (like rocks in a cobblestone road) to synthesize large non-repeating arrangements of elements [7]. Sketch-based synthesis enables interactive sketching with discrete element textures [9]. Repetitive structures can also be synthesized with example-based methods [31]. Sketch-based interfaces have also been used to design and guide dynamic fluid simulations [32] and for sketching crowds of agents [33]. Finally, sketch-based interaction has been used in VR to paint in virtual space [34]. We build on discrete element synthesis and sketch-based interaction for creating and configuring agents in a 3D VR simulation.

III. Sketch-based synthesis of Agents

In this section, we describe our method for generating agents through sketch-based synthesis. As shown in Figure 2, we use sketch based synthesis to interactively design, configure and guide the agents in a live simulation. Our objective is to make simulations interactive experiences. With our approach, a user can rapidly try out new ideas and experiment with agents; we keep the human in the loop.

Sketch-based synthesis generates agents along the brush path relative to examples created in an example palette. The spatial arrangement, behaviors and other properties of the agents are chosen from the example to minimize differences. A relaxation procedure further improves the synthesized agents to match the example closely. It is important to point out that the arrangements we synthesize are not just stochastic, they can be highly organized, such as the lipids in our mitochondrion (Figure 1).

We use VR to put the user inside of the simulation. VR gives the user depth perception and the ability to walk around and interact with a live agent-based simulation. These factors increase immersion. We leverage VR input devices as a natural and intuitive way to paint agents directly in 3D.

Sketch-based synthesis

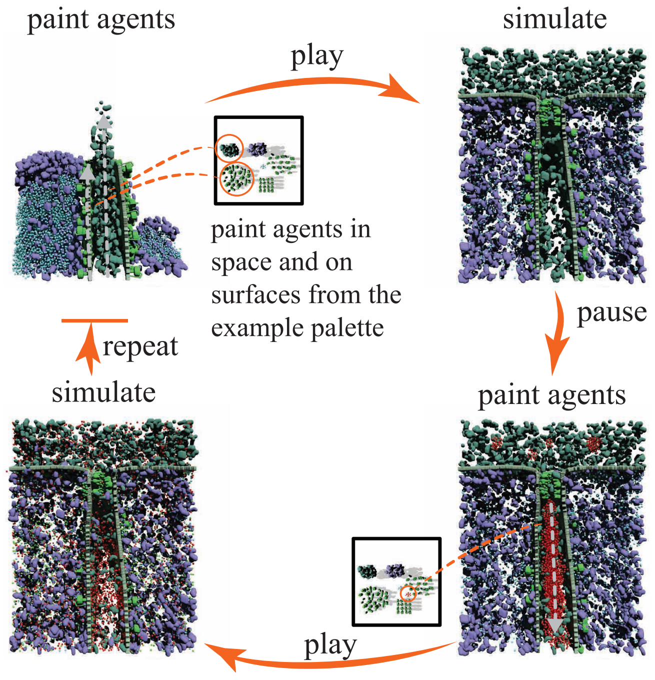

paint agents
play
simulate
paint agents in space and on surfaces from the example palette
pause
repeat
simulate
paint agents
play

Figure 3. Our system enables **iterative design and simulation**. We paint a simulation, press play and simulate. Then we pause, paint new agents and repeat. The example palette enables us to create new agent arrangements and configurations to paint into our simulations.

A generative brush creates agents along the brush path as we move it through space or along a surface (Figure 2b). We use an example-based discrete element texture synthesis algorithm [9] that builds on work by Ma et al. [7] and Ijiri et al. [8]. With example-based synthesis, we give an algorithm a small example arrangement of agents. An algorithm then synthesizes new agents along the brush path so that the synthesized agents have a similar, but not repeating, arrangement and assignment of agent properties relative to the example. This is a 'what you see is what you get' method for defining agent configurations and arrangements.

Revisiting our mitochondrion example, various regions and surfaces can be filled and covered with different types and arrangements of agents (Figure 2). The example palette is a space where the user designs arrangements of agents and configures their behavior and other properties (Figure 2a). 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 (Figure 2). See [9] for a more detailed description of this algorithm.

We can design the palette manually, but we can also use our generative tools to rapidly sketch new agent arrangements, apply agent behaviors and properties from another example, and manually tweak the parameters of some of the agents. Examples in the palette can be stochastic (Figure 8b) or structured (Figure 8a). To specify a spatial relationship the user only has to place agents near each other. The synthesis algorithm will use the spatial arrangement as an example of what to synthesize.

Discrete element synthesis

A discrete element is a particle with a position, radius and attributes vector [7, 8]. For example, an attribute vector might contain the shape and color of an element.

The goal of discrete element texture synthesis is to create large-scale textures that are not repetitive, such as the membrane proteins in our mitochondrion (Figure 1). 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 [35]. In our discrete element texture synthesis algorithm, a neighborhood cost function (based on [7]) determines the degree of similarity between windows in an example and output discrete element texture. The goal of discrete element texture synthesis is to minimize the local neighborhood cost everywhere in the output relative to matching neighborhoods in the example.

In our system, we measure the similarity of two neighborhoods of elements by aligning them with each other and finding pairs of elements (Figure 4). The differences between elements in a pair are summed with the other pairs of elements to give the similarity between the two neighborhoods.

Let a be an element, it has a position p_a, radius r_a and attributes α_a. A neighbourhood of radius r_a around a is given by $n(a)$. An element in the neighbourhood of a is given by $a' \in n(a)$. We use Ma et al.'s definition of neighbourhood distance [7]. The neighbourhood distance

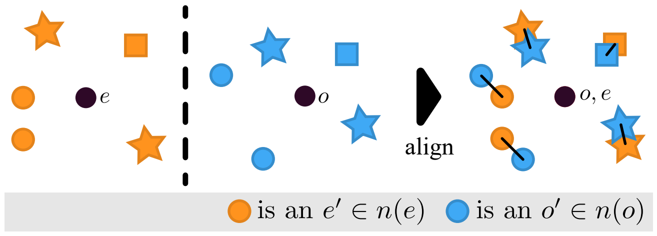


Figure 4. To compare two neighborhoods of agents $n(e)$ and $n(o)$, we align the two neighborhoods and compare pairs of elements. The difference in position and attributes between elements in a pair determine the similarity of the two neighborhoods.

between a and another element b is:

$$|n(a)-n(b)| = \sum_{a' \in n(a)} \|(p_{a'}-p_a)-(p_{b'}-p_b)\| + \omega(\alpha_{a'}, \alpha_{b'}). \tag{1}$$

where a' is an element in the neighbourhood $n(a)$ and the matching pair for that element in $n(b)$ is denoted with b'. Pairs are found by aligning the two neighbourhoods and greedily assigning elements from $n(a)$ to $n(b)$ that are nearby (see Davison et al. [9] for details). $\omega(a, b) \in \mathbb{R}$ is a customizable function that compares the attribute vectors of two elements. For example, in our mitochondrion simulation, we simply use the dot product, e.g., $\omega(\alpha_{a'}, \alpha_{b'}) = \alpha_{a'} \cdot \alpha_{b'}$. We will discuss the attribute vector and how we use it to represent the properties and behaviors of an agent in the next section.

Agent synthesis

A general definition of an agent is as a 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 [36]. From our perspective, an agent is a software object with a set of properties and behaviors that determine the set of situations it can be in, its actions, internal data and decision function.

Our element-synthesis framework is modular and separate from the implementation of the agent simulation. Internally, we synthesize discrete elements. Then, just before we simulate, we take the synthesized elements and convert them into agents. When we pause and start painting again, we convert the simulation agents back into elements. The user creates and defines agents in the example palette, and then we convert them into elements for use by our element-synthesis framework.

Elements are representations of agents in our framework. Our idea is to map the properties and behaviors of agents into the component vectors of elements (Figure 5), where they can be compared with a neighborhood similarity function (Equation 1). Anything that we can't put into the element attribute vector, we attach to an additional data property on the element.

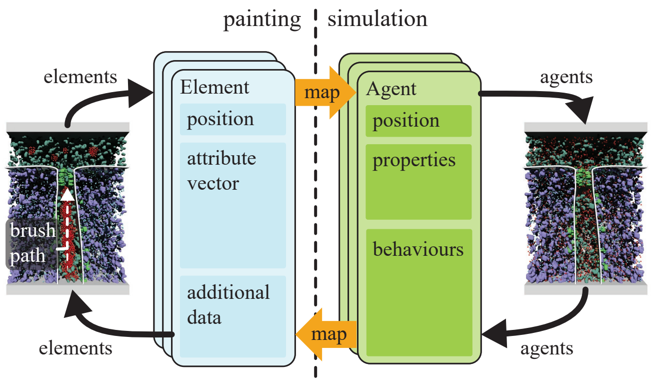


Figure 5. Our generative tools synthesize elements, when we simulate, we map elements onto agents and vice versa. When we paint, we take the simulation agents and map them onto elements. Elements and agents share positions. The properties and behaviors of agents are mapped onto element attribute vectors. Those things that can't be mapped into the attribute vector go into the additional data component of an element, where they have no impact on the element cost function.

Hence, we maintain two domains: the agent domain $agents$ and the element domain $elements$. We use our sketch-based tools on $elements$, and run our simulations on $agents$. The simulation designer defines a map M : $agents \longleftrightarrow elements$ that is used to convert the agent domain to an element domain, and vice vera. As an example, let us have a look at how we map an ATP synthase agent to an element (Figure 6). The ATP synthase behavior class is part of the attribute vector, along with the attributes that determine the rate at which adenosine diphosphate is converted to adenosine triphosphate, the color and shape of the agent, and the rate at which it spins. The remaining internal data is added to the additional data property.

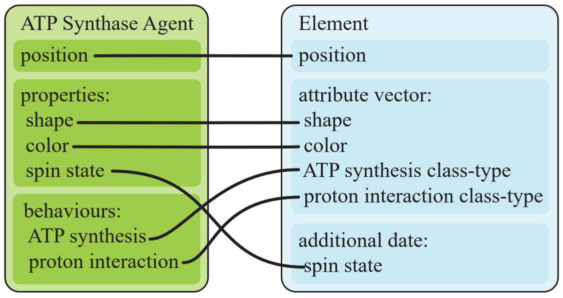


Figure 6. Here we give an example of mapping an ATP Synthase agent to an element. We copy the position directly between the two. The shape and color are added to the attribute vector. The ATP synthesis and proton interaction behaviors are mapped as class-types in the attribute vector. We don't map the spin state of the agent onto the attribute vector, so we attach it to the element's additional data area.

We make a distinction between agents and elements for two reasons: (1) For interactive sketch-based synthesis, efficiency is critical. Therefore, elements are essentially a vector that can be efficiently copied and stored, whereas data structures for an agent can be significantly more complicated. Our element framework has internal data structures to accelerate queries based on the element vector representation. (2) By converting an agent to an element, the simulation designer controls what agent attributes our system can create and modify.

IV. Sketch-Based Synthesis and Simulation in Virtual Reality

Sketch-based interaction is inspired by physical paint-and-canvas interactions and has been extensively explored for 3D modeling [24]. These tools can be used to design the environment of an agent-based simulation, such as the geometry of the mitochondrial membranes (Figure 1). Our contribution focuses on the agents in a simulation and a new set of sketch-based tools to synthesize and configure agents in VR.

Virtual reality

We use VR controllers as a painting tool in a 3D simulation. Using Unreal Engine 4 [37], we implemented VR based generative brushes that paint elements along the 3D path of positionally-tracked VR controllers. The user holds two controllers (Figure 7). Attached to the left controller is a user interface. The right controller is used to interact with the interface and for sketch-based interaction.

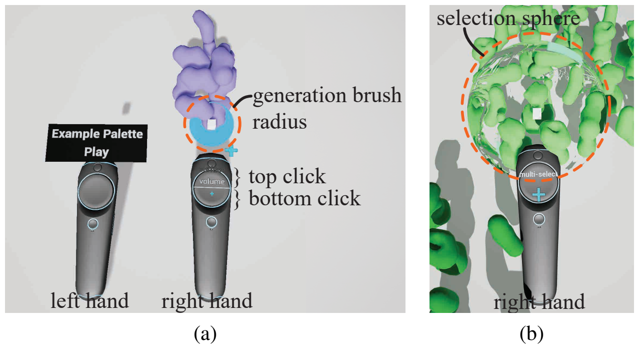


Figure 7. (a) The **generation** user interface on the left and right controller. The user can select controls on the left hand with the right hand to show the example palette and play the simulation. Clicking the top of the touch-pad with the thumb toggles surface and volume painting, while clicking the bottom toggles adding or erasing elements. The current brush radius is indicated by a sphere and controlled by the trigger button. (b) The **example palette selection** interface has a selection sphere and a touch-pad toggle for adding or removing elements from the selection.

We support room-scale VR navigation (Figure 2) and interactive navigation gestures. **Teleportation** involves pointing at a location and pressing a button to teleport there. Like an astronaut pulling his/her way through a space station, the **grab** gesture can be used to pull oneself through the world.

We embed the agent-based simulation inside a box and the **stretch** gesture uses two hands to resize and rotate that box.

Sketch-based synthesis: VR interfaces and tools

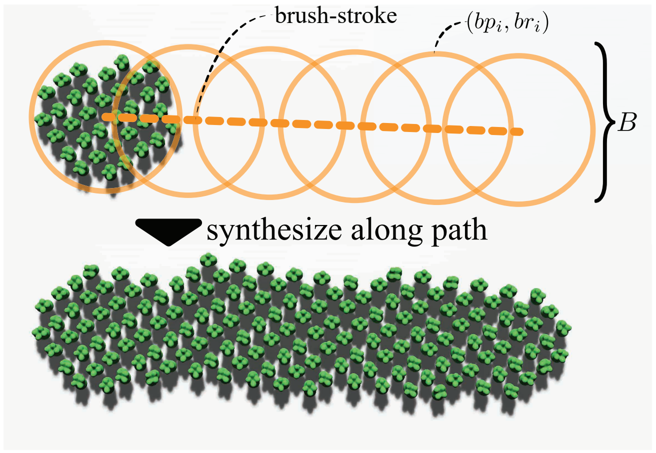
The **generative-brush** (Figure 8a) synthesizes new elements in a small region along the 3D brush path using interactive discrete element synthesis [9]. The path B is composed of a set of brush-spheres which have a position and a radius (bp_i, br_i). We set the radius of the brush sphere with the VR controller trigger button.

The generative-brush synthesizes new elements, but 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 behavior to agents in a scene that did not have this behavior before (Figure 11).

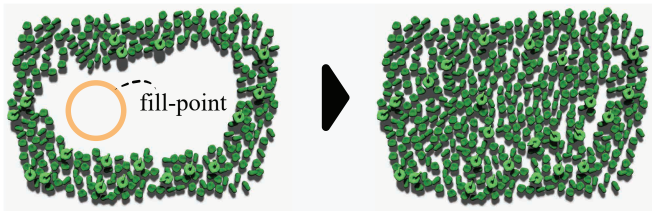
With the **filler tool,** (Figure 8b) the user identifies a fill point where there are no elements, then synthesized elements are added until there is no more room to do so. The **eraser** (Figure 8c) removes elements within a certain distance along a brush path. The **selection brush** selects agents in a radius around the brush.

Example palette construction

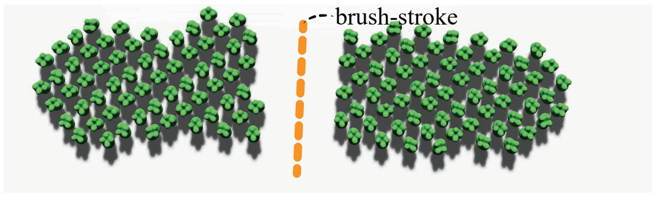
The user builds the example palette with three different methods: (1) manual placement and configuration of agents, (2) copy-paste operations, and (3) use of the generative brush. Our implementation is based on the Unreal Engine 4 Editor ([37]). Therefore we leverage the Unreal's user interface for manually placing agents in 3D space and configuration through Unreal's property editor windows. The generative brush can be used directly inside of our VR environment to paint new examples. For the copy-paste operation, the user selects agents with the selection brush from the palette or the simulation, and then the user specifies a target location in the example palette to copy the agents too.



(a) **Generative-Brush** A randomly selected patch from the example palette selection is copied to the output at the first brush point to seed region-growing along the rest of the brush points in B.



(b) **Filler Tool** The empty region is filled with elements starting at the fill-point.



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

Figure 8. Our sketch-based tools applied to a planar surface.

V. An example LifeBrush simulation

To illustrate our iterative sketch-based synthesis and our agent-based system we go back to our mitochondrion example (Figure 1). We describe an example interaction with our sketch-based tools in Figures 9, 10, 11 and 12 (see https://youtu.be/HYLvN2qijeA for a video of this interaction).

The mitochondrion is an organelle in Eukaryotic cells (cells with a nucleus) that generates most of the cell's adenosine triphosphate (ATP), a source of chemical energy. ATP-synthase molecules in the inner mitochondrial membrane between the cristae and matrix space (Figure 1), combine phosphate and adenosine diphosphate (ADP). The activity of ATP-synthase is driven by a proton gradient between the cristae and matrix space when a gradient is present, protons flow through ATP-synthase, causing it to mechanically spin and change its shape in such a way that bound ADP and phosphate molecules are combined to form ATP. The proton gradient is maintained by another set of chemical reactions that drive protons through pumps from matrix space into the cristae.

Implementation details and performance

Our framework for sketch-based synthesis and agent-based simulation has two main components: (1) the sketch-based element synthesis framework and (2) an agent-based software interface to define the mapping function for converting between agents and elements.

Our framework is implemented in the Unreal Engine 4 game engine [37]. Therefore, the simulation designer can use any of the tools provided by this engine to define their agent-based simulations and Unreal's property editor windows to

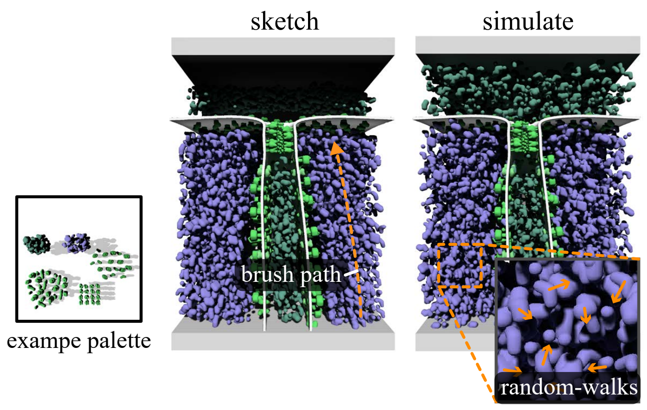


Figure 9. Sketching the initial state of a mitochondrion simulation. We painted the agents in this simulation from the example palette. At this point the agents only have a random-walk behavior and the simulation models an inactive mitochondrion.

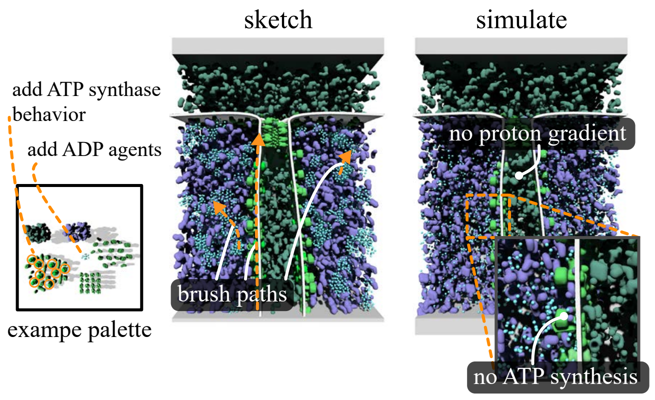


Figure 10. ATP synthase combines phosphate with adenosine diphosphate (ADP) to create adenosine triphosphate (ATP). In the Unreal Engine interface, we add this behavior to the ATP synthase agents in the example palette. Then we paint the behavior onto the ATP synthase agents in our simulation and add ADP to the simulation. When we simulate, ADP binds to ATP synthase. However, we need a proton gradient to get ATP synthase spinning and produce ATP.

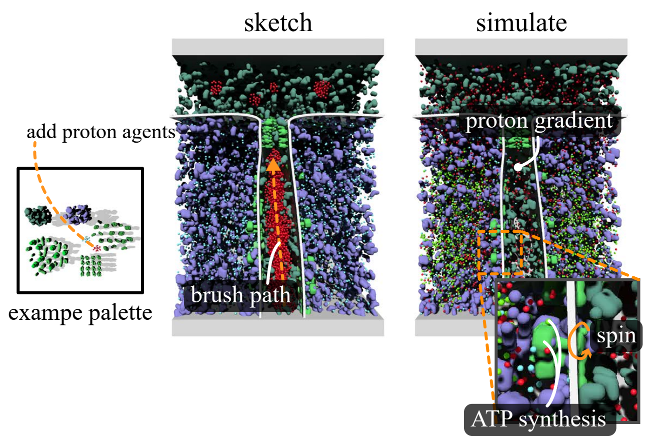


Figure 11. Here we add proton agents (that represent a large number of protons) to the example palette and paint a proton gradient into the mitochondrion's cristae (central region in this figure). When we simulate, the proton gradient drives protons through ATP synthase, causing it to spin and produce ATP.

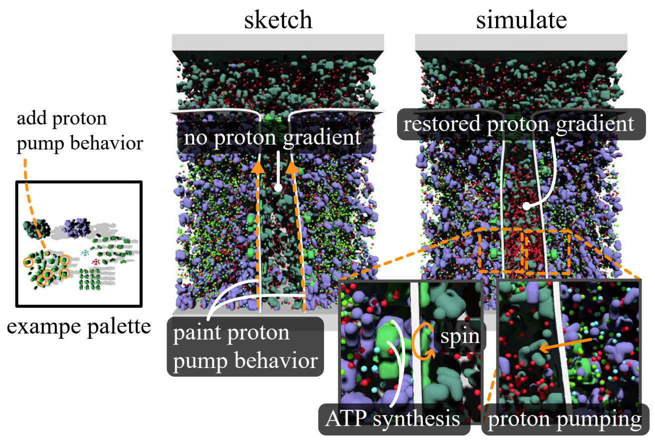


Figure 12. Eventually the proton gradient equalizes, and ATP synthase stops producing ATP. In this step, we add a proton pump behavior to some of the agents in the example palette and paint that new behavior onto the proton pumps in our simulation. We simulate, and a proton gradient is restored. Eventually, ATP synthase starts spinning again and producing ATP.

configure their agents in the example palette.

We developed a simulation environment suitable for simulating large numbers of agents in virtual reality. Agents are efficiently rendered using Unreal's `UInstancedStaticMeshComponent` class. We use Flex, a GPU based particle-physics engine to handle the rigid body collisions and forces between our agents [38, 39].

We ran the simulation on an Intel 5960x processor with eight cores (3.0 GHz), 16 GB of RAM and a Nvidia GTX 1080 GPU. Our simulation runs at 90 frames-per-second with about 10,000 agents.

VI. Discussion and Future Work

We used an example palette of agent arrangments to paint a mitochondrion simulation with our VR sketch-based tools. We showed how our tools could be used to experiment and interact with this simulation. We also demonstrated interactive simulation in virtual reality. Our simulation is a simple illustration of what is possible with sketch-based synthesis and agent-based simulation. In future work, we will look at creating and evaluating more complicated and detailed simulations. We imagine applications of our method for interactive illustration and teaching.

We implemented a region-growing algorithm for discrete element texture synthesis. However, there are other techniques for statistical synthesis that we would like to apply to agent-synthesis, such as [31, 40].

VR has raised some interesting interaction and visualization challenges with densely populated scenes that need to be explored. Our approach is limited by the number of agents that can be simulated and rendered in real-time within VR. Future research could address these limitations on different fronts, for example by using automatic abstraction to reduce complexity (similar to Shirazi et al. [23]), multi-scale modeling and simulation, and levels-of-detail rendering techniques.

References

[1] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *ACM SIGGRAPH computer graphics*, vol. 21, no. 4. ACM, 1987, pp. 25–34.

[2] A. Esmaeili, T. Davison, A. Wu, J. Alcantara, and C. Jacob, "Prokaryo: an illustrative and interactive computational model of the lactose operon in the bacterium escherichia coli," *BMC bioinformatics*, vol. 16, no. 1, p. 311, 2015.

[3] D. S. Goodsell, *The machinery of life*. Springer Science & Business Media, 2009.

[4] M. Zick, R. Rabl, and A. S. Reichert, "Cristae formation—linking ultrastructure and function of mitochondria," *Biochimica et Biophysica Acta (BBA)-Molecular Cell Research*, vol. 1793, no. 1, pp. 5–19, 2009.

[5] T. Klein, L. Autin, B. Kozlíková, D. S. Goodsell, A. Olson, M. E. Gröller, and I. Viola, "Instant construction and visualization of crowded biological environments," *IEEE transactions on visualization and computer graphics*, vol. 24, no. 1, pp. 862–872, 2018.

[6] D. Yuen and C. Jacob, "Eukaryo: An agent-based, interactive simulation of a eukaryotic cell," in *Artificial Life Conference 2016*, 2016, p. 562.

[7] C. Ma, L.-Y. Wei, and X. Tong, "Discrete element textures," in *ACM Transactions on Graphics (TOG)*, vol. 30, no. 4. ACM, 2011, p. 62.

[8] T. Ijiri, R. Mech, T. Igarashi, and G. Miller, "An example-based procedural system for element arrangement," in *Computer Graphics Forum*, vol. 27, no. 2. Wiley Online Library, 2008, pp. 429–436.

[9] T. Davison, F. Samavati, and C. Jacob, "Interactive example-palettes for discrete element texture synthesis," University of Calgary, Department of Computer Science, Canada, Tech. Rep., 04 2018.

[10] Harvard BioVisions, "The inner life of the cell (video)," 2007.

[11] Z. Lv, A. Tek, F. Da Silva, C. Empereur-Mot, M. Chavent, and M. Baaden, "Game on, science-how video game technology may help biologists tackle visualization challenges," *PloS one*, vol. 8, no. 3, p. e57990, 2013.

[12] M. Norrby, C. Grebner, J. Eriksson, and J. Boström, "Molecular rift: virtual reality for drug designers," *Journal of chemical information and modeling*, vol. 55, no. 11, pp. 2475–2484, 2015.

[13] B. Kozlikova, M. Krone, N. Lindow, M. Falk, M. Baaden, D. Baum, I. Viola, J. Parulek, H.-C. Hege *et al.*, "Visualization of biomolecular structures: state of the art," in *Eurographics Conference on Visualization (EuroVis)-STARs*. The Eurographics Association, 2015, pp. 061–081.

[14] L. Martínez, R. Andrade, E. G. Birgin, and J. M. Martínez, "Packmol: a package for building initial configurations for molecular dynamics simulations," *Journal of computational chemistry*, vol. 30, no. 13, pp. 2157–2164, 2009.

[15] G. T. Johnson, L. Autin, M. Al-Alusi, D. S. Goodsell, M. F. Sanner, and A. J. Olson, "cellpack: a virtual mesoscope to model and visualize structural systems biology," *Nature methods*, vol. 12, no. 1, p. 85, 2015.

[16] J. W. Haefner, *Modeling biological systems: principles and applications*. Springer Science & Business Media, 2012.

[17] I. Burleigh *et al.*, "Biomolecular swarms–an agent-based model of the lactose operon," *Natural Computing*, vol. 3, no. 4, pp. 361–376, 2004.

[18] C. Jacob, A. Barbasiewicz, and G. Tsui, "Swarms and genes: Exploring λ-switch gene regulation through swarm intelligence," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. IEEE, 2006, pp. 2535–2542.

[19] V. Sarpe and C. Jacob, "Simulating the decentralized processes of the human immune system in a virtual anatomy model," in *BMC bioinformatics*, vol. 14, no. 6. BioMed Central, 2013, p. S2.

[20] J. R. Karr, J. C. Sanghvi, D. N. Macklin, M. V. Gutschow, J. M. Jacobs, B. Bolival Jr, N. Assad-Garcia, J. I. Glass, and M. W. Covert, "A whole-cell computational model predicts phenotype from genotype," *Cell*, vol. 150, no. 2, pp. 389–401, 2012.

[21] C. Jacob, S. von Mammen, T. Davison, A. Sarraf-Shirazi, V. Sarpe, A. Esmaeili, D. Phillips, I. Yazdanbod, S. Novakowski, S. Steil *et al.*, "Lindsay virtual human: Multi-scale, agent-based, and interactive," in *Advances in Intelligent Modelling and Simulation*. Springer, 2012, pp. 327–349.

[22] A. Wu, T. Davison, and C. Jacob, "A 3d multiscale model of chemotaxis in bacteria," in *Artificial Life Conference 2016*, 2016, p. 546.

[23] A. S. Shirazi, T. Davison, S. von Mammen, J. Denzinger, and C. Jacob, "Adaptive agent abstractions to speed up spatial agent-based simulations," *Simulation Modelling Practice and Theory*, vol. 40, pp. 144–160, 2014.

[24] L. Olsen, F. F. Samavati, M. C. Sousa, and J. A. Jorge, "Sketch-based modeling: A survey," *Computers & Graphics*, vol. 33, no. 1, pp. 85–103, 2009.

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

[26] P. Merrell and D. Manocha, "Model synthesis: a general procedural modeling algorithm," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 6, pp. 715–728, 2011.

[27] A. Emilien, U. Vimont, M.-P. Cani, P. Poulin, and B. Benes, "Worldbrush: Interactive example-based synthesis of procedural virtual worlds," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 106:1–106:11, Jul. 2015.

[28] J. Gain, H. Long, G. Cordonnier, and M.-P. Cani, "Ecobrush: Interactive control of visually consistent large-scale ecosystems," in *Computer Graphics Forum*, vol. 36, no. 2. Wiley Online Library, 2017, pp. 63–73.

[29] K. Ketabchi, A. Runions, and F. F. Samavati, "3d maquetter: Sketch-based 3d content modeling for digital earth," in *2015 International Conference on Cyberworlds (CW)*, Oct 2015, pp. 98–106.

[30] F. Samavati and A. Runions, "Interactive 3d content modeling for digital earth," *The Visual Computer*, vol. 32, no. 10, pp. 1293–1309, 2016.

[31] R. Roveri, A. C. Öztireli, S. Martin, B. Solenthaler, and M. Gross, "Example based repetitive structure synthesis," in *Computer Graphics Forum*, vol. 34, no. 5. Wiley Online Library, 2015, pp. 39–52.

[32] B. Zhu, M. Iwata, R. Haraguchi, T. Ashihara, N. Umetani, T. Igarashi, and K. Nakazawa, "Sketch-based dynamic illustration of fluid systems," in *ACM Transactions on Graphics (TOG)*, vol. 30, no. 6. ACM, 2011, p. 134.

[33] Q. Gu and Z. Deng, "Formation sketching: an approach to stylize groups in crowd simulation," in *Proceedings of Graphics Interface 2011*. Canadian Human-Computer Communications Society, 2011, pp. 1–8.

[34] Google LLC, "Tilt brush software," 2016.

[35] L.-Y. Wei, S. Lefebvre, V. Kwatra, and G. Turk, "State of the art in example-based texture synthesis," in *Eurographics 2009, State of the Art Report, EG-STAR*. Eurographics Association, 2009, pp. 93–117.

[36] M. Afsharchi, B. H. Far, and J. Denzinger, "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, pp. 923–930.

[37] Epic Games, Inc., "Unreal engine 4 game engine," 2018.

[38] Nvidia Corporation, "Flex particle physics library," 2018.

[39] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, "Unified particle physics for real-time applications," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 104, 2014.

[40] R. Roveri, A. C. Öztireli, and M. Gross, "General point sampling with adaptive density and correlations," in *Computer Graphics Forum*, vol. 36, no. 2. Wiley Online Library, 2017, pp. 107–117.