Generating Molecular Motion Blur Videos for a User Study

CS Report

Johannes Eschner

August 2022

1 Introduction

This report documents the process of creating a series of short animated videos of a molecular scene. The purpose of these videos is to conduct a user study, which investigates whether motion blur effects can be used as an illustrative tool to better guide the user's attention in crowded visualizations of complex biological processes.

The aim of the implementation is to use an existing framework to create the necessary videos and to iteratively explore different approaches to achieve the desired motion blur effect. As the expected outcome is a series of non-interactive videos, there is no emphasis on delivering the results in real-time. While the molecular rendering framework we use is capable of rendering animations in real-time, our motion blur implementation is a pure post-processing effect, which is applied to videos and not to interactive visualizations.

In the rest of the report there are two main sections detailing the implementation of the video generation. First there is a section which explains the molecular rendering, including the scene configuration and the reaction animation. The second section provides details on the motion blur post-processing and the automated video generation.

2 Implementation

2.1 Molecular Rendering

The molecular renderer we utilize for generating the videos is the molecular visualization framework Marion [2]. As our study scene shows a simple reaction which consists of two molecules briefly attaching and then detaching again, the scene is entirely constructed and does not represent an actual chemical reaction. All molecules are alpha-beta tubulin proteins, rendered in an atom-level representation.

2.1.1 Configuration

The scene which is used for the user study consists of a total of 1000 molecules, of which two will react during the animation. Each video is characterized by the parameters seen in Table 1.

For the study we created a total of 40 videos, varying the parameters speed (4 levels), trail length (5 levels), smoothness (2 levels) automatically. Each randomized parameter (see Table 1) takes the video index (0 to 39) as seed value to be able to reproduce single videos with with same settings. The configuration of each video is encoded in the file name, which has the following structure `012_tl1_ts_cs_s2_170_190`. The first number in the filename is the video ID, which is used as random seed. The value after `tl` is the trail length, `cs` stands for the

sparse scene (1000 molecules) whereas `cd` would signify a dense scene with 5000 molecules. This dense scene, however, is not part of the final study. `s2` shows that the video has a speed level of 2. Finally, the numbers `170` and `190` are the indices of the two reactants, which are selected by drawing two random numbers (a, b with $a \not\equiv b \pmod 8$) between 0 and the number of molecules in the scene. In the reactant IDs the color of the two molecules in the video is also implicitly encoded as $ID \pmod 8 = ColorIndex$.

Pre-Defined Parameters (from config)		
Parameter Name	Possible Values	Description
Speed	[2, 8]	The movement speed of individual molecules in the scene. Speed level 2 corresponds to an average speed of 480nm/s. Increases with a step size of 2 to a maximum of 8 which corresponds to an average speed of 1.92μm/s. For reference a single alpha-beta tubulin protein has a length of 8nm [1].
Smoothness	[0.0, 1.0]	Determines the strength of the noise applied to the molecule motion paths. Only applies to the context molecules and not the reaction. 0 corresponds to full noise level while 1 removes the second pass of the noise function.
Trail Length	[0, 4]	Length level of the motion blur trail within screen space. 0 means no motion blur. Level roughly corresponds to trail length in molecule lengths, depending on random speed variation.
Random Parameters		
Parameter Name	Possible Values	Description
Reactant One	[0, 999]	Index of the first reactant molecule in the protein instance buffer. Must fulfil: $ReactantOne \not\equiv ReactantTwo \pmod 8$ (to have two different colors)
Reactant Two	[0, 999]	Index of the second reactant molecule in the protein instance buffer. Must fulfil: $ReactantOne \not\equiv ReactantTwo \pmod 8$ (to have two different colors)
Reaction Timestamp	[0.25, 0.75]	Timestamp at which the two reactants first attach. Has to be within the middle 50% of the animation.
Position Seed	[0, 999]	Random seed for the position at which the reactants react in the animation.

Table 1: Parameters by which the scene is defined. A decimal point in the Possible Values column represents a floating point value, otherwise values are integers. All intervals are closed intervals.

2.1.2 Scene

In the Marion framework there is a scene called `dots.cpp` which contains the code for setting up and rendering the molecular scene. Here, the scene parameters such as the total amount of molecules in the scene (`numPoints`) are defined. The parameters which are defined in the config file (`/apps/vr4d/work/scripts/dots/config.json`) of the scene are also collected within the scene file and passed to the shader. The animation of the molecules takes place in the vertex shader of the app (`/apps/vr4d/work/shaders/rendering/cellVIEW/proteins/renderproteins.vs`).

2.1.3 Vertex Shader

The molecular animation takes place in the vertex shader. Protein instances are passed into the shader using buffers, which contain information on their position and IDs. The scene parameters, which determine the characteristics of the animation, are passed into the shader as uniforms. For each of the reactant molecules an ID is passed to the shader. This ID is the index at which the reactants lie in the protein instance buffer.

The molecular motion is animated using two pseudo-random noise functions, one for translation and one for rotation. The base noise function is based on an implementation from Shadertoy [1]. At each frame in the animation, which for the study has a duration of 20 seconds, the next iteration of the continuous random noise function determines the new position and rotation of each protein. To account for the randomness of Brownian motion, the random position value is calculated by a weighted sum of two offset random noise iterations, where the weight of the second iteration is the smoothness parameter ranging between 0 and 1. Using this smoothness parameter the jitter of the proteins is adjusted. Here, there is a distinction between the context molecules in the background and the two reactants. While the reactants are always shown with unsmoothed motion, the smoothness of the context molecules can be adjusted using the smoothness parameter.

Apart from the definition of the two reactant indices, the reaction animation is controlled by two more randomised parameters. The first parameter is the reaction timestamp, which for the 20 second animation for the study lies between 5 and 15 seconds into the animation (see Figure 1). Note that internally the 20 second animation is represented as a time span between 0 and 1. The second parameter is a seed value which determines the random position of the reaction using the same pseudo-random noise function as the molecular movement. In the reaction animation the two reactants will initially behave like the context molecules, moving around on random paths. Five seconds before the reaction timestamp the random trajectories of the two reactants start a linear interpolation towards the random reaction position. This linear interpolation between the random motion and the linear trajectory towards the reaction position is based on the work of Le Muzic et al. [3].

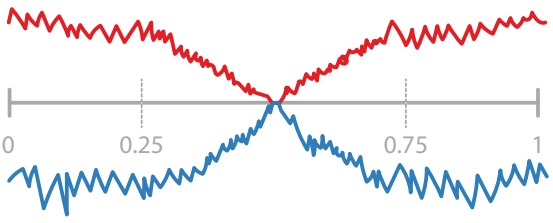


Figure 1: Schematic representation of a reaction between two reactants (red and blue). The reaction timestamp is approximately at 0.5 with the reaction interpolation starting at 0.25 and ending at 0.75.

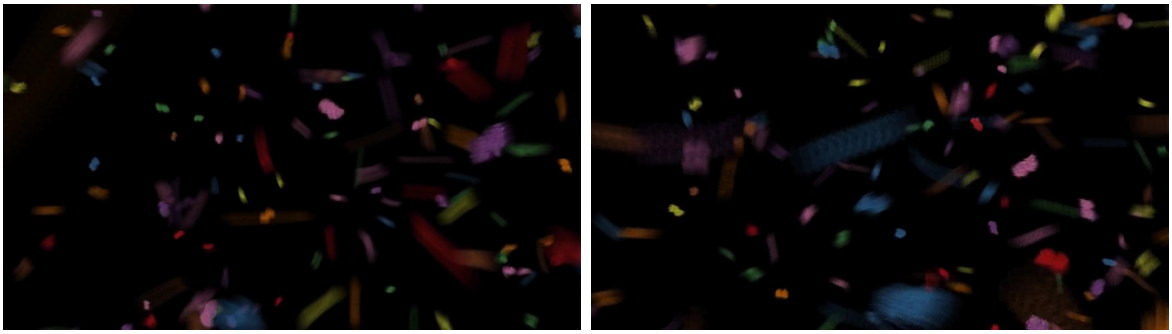
[1] https://www.shadertoy.com/view/4dS3Wd

Apart from the animation the vertex shader is also responsible for assigning a random color to each molecule. This is done by choosing one of eight colors from the qualitative color set '8-class Set 1' from ColorBrewer [2] based on the mod of the protein instance ID.

2.2 Post-Processing and Video Generation

For the simple videos without any motion blur added, the scene is rendered straight to PNG frames, each containing the reactants and the context molecules. In videos where motion blur is to be applied, the scene is rendered twice. First only the reactants are rendered, discarding all the fragments which only contain context molecules in the fragment shader. Then the context molecules are rendered with the reactant molecule fragments set to black to mask out any occlusions. This distinction has to be made as we only want to apply the motion blur trails to the context and not the reactants. Masking in the other direction is not necessary as the blurred motion blur trails will be semi-transparent after the post-processing step and therefore they will never fully occlude the reactants. With the PNG frames generated, we use FFmpeg [3] to compile them into video files. As we need a high temporal resolution in order to achieve smooth motion blur trails, we render the scenes with 120 frames per second.

Once each version of the scene is rendered, the motion blur trails are added to the context molecules using a Python script which accumulates a set number of frames and then averages their color intensities. The number of frames to be accumulated depends on two parameters. One is the speed of the given video as determined by the scene parameters and the other is the trail length which determines the absolute length of a trail in screen-space independent of the speed. This way we can control the factor trail length independently of the speed. All accumulated averaged frames are then saved into a new video sequence which now contains the final motion blur trails. An example of this can be seen in Figure 2.



(a) Scene with speed setting 2 and trail length 4. **(b)** Scene with speed setting 8 and trail length 4.

Figure 2: Two different speeds with motion blur trails of the same length applied to the outputs.

With the trails generated the last step in the post-processing pipeline is to generate the final videos by blending the motion blurred context molecules with the separate reaction videos. This is done by using the FFmpeg filter library and employing the *Screen* blend mode. All post-processing is performed by a single Bash script which, when placed in the output directory, generates the final video files from the PNG frames generated by Marion.

A possible future improvement on this video generation approach would be to implement the motion blur trail generation in Marion. This way the post-processing could be done in real-time.

[2] https://colorbrewer2.org/#type=qualitative&scheme=Set1&n=8
[3] https://ffmpeg.org/

References

[1] Shinya Inoué and Edward D. Salmon. "Force Generation by Microtubule Assembly/Disassembly in Mitosis and Related Movements". In: *Molecular Biology of the Cell* 6.12 (1995). PMID: 8590794, pp. 1619–1640. DOI: 10.1091/mbc.6.12.1619. eprint: https://doi.org/10.1091/mbc.6.12.1619.

[2] Peter Mindek, David Kouřil, Johannes Sorger, David Toloudis, Blair Lyons, Graham Johnson, Meister Eduard Gröller, and Ivan Viola. "Visualization Multi-Pipeline for Communicating Biology". In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (2017).

[3] Mathieu Le Muzic, Julius Parulek, Anne-Kristin Stavrum, and Ivan Viola. "Illustrative Visualization of Molecular Reactions using Omniscient Intelligence and Passive Agents". In: *Computer Graphics Forum* 33.3 (June 2014). Article first published online: 12 JUL 2014, pp. 141–150.