Review: A Web-Based Simulation Viewer for Sharing Evolutionary Robotics Results

Anthony J. Clark
Computer Science Department
Missouri State University
Springfield, Missouri, USA
anthonyclark@missouristate.edu

Jared M. Moore
School of Computing and Information Systems
Grand Valley State University
Allendale, Michigan, USA
moorejar@gvsu.edu

ABSTRACT

Evolutionary robotics researchers often need to share results that may be too difficult to describe in text and too complex to show using images. Many researchers include links to videos as supplementary materials, but videos have a predefined view of the scene and do not allow watchers to adjust the viewing angle to their preference. In this paper we present a web-based application (based on three.js) for sharing interactive animations. Specifically, our tool (called Review) enables researchers to generate simple animation log data that can be loaded in any modern web browser on a computer or mobile device. The camera in these animations is controlled by the user such that they can pan, tilt, rotate, and zoom in and out of the scene. Review is meant to improve the ability of researchers to share their evolved results with one another.

CCS CONCEPTS

- **Computing methodologies** → **Scientific visualization**; *Evolutionary robotics*;

KEYWORDS

evolutionary robotics, visualization

ACM Reference Format:
Anthony J. Clark and Jared M. Moore. 2018. Review: A Web-Based Simulation Viewer for Sharing Evolutionary Robotics Results. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3205651.3208292

1 INTRODUCTION

Conveying the behaviors of a dynamic system is extremely important in the field of robotics. When presenting the operation of a new robotic system, it is vital to provide readers and collaborators with some way to visualize its motions. This is particularly true for evolutionary robotics (ER) research, as often the goal of ER is to produce *novel* or unexpected behaviors. Despite this importance, however, it can be difficult to present behaviors in a manner useful to other researchers. In this paper, we present **Review**[1] (source code[2]), a web-based platform for sharing visualizations of evolved robotic systems.

Traditionally, to depict behaviors of an evolved system, researchers provide sequences of images (see Figures 1 and 2) and/or links to videos. Image sequences have been a common feature of ER from the beginning; see, Sims [14] from 1994 for example. Images, however, can sometimes be problematic. First, when a reader is unfamiliar with terminology (e.g., gait, hopping, bounding, etc.) they may not have a baseline reference with which to compare and understand images of a new behavior. And second, even with a good understanding of what to expect, some complex movements are difficult to comprehend from image sequences. For example, the quadruped gaits in Figure 1 are reasonably understandable for someone familiar with such systems, but for someone new to quadrupedal robotics it may be difficult to picture the motions. Likewise, the unusual gaits exhibited by the worm-like animat in Figure 2 are difficult to imagine without the aid of video (or animation).

Linking to videos is a common approach to handling the problems associated with images. Videos, however, are similarly static in the sense that the camera transform (viewing angles, zoom levels, etc.), object materials (colors and other material properties), and playback speed are fixed once the video is generated and uploaded. Practically, this means that a viewer of the video cannot change the view angle or zoom-in on different aspects of the scene, and cannot change the color of objects if they find the chosen colors hard to see. And although some web-based video players allow a viewer to speed-up/slow-down the video by small amounts, this is usually at the cost of playback smoothness; videos appear *choppy* when they are slowed down since videos are recorded at a fixed frame-rate and slowing down the video effectively makes each frame appear for a longer period of time. Additionally, it is common for ER researchers to generate videos by performing a video screen recording (often referred to as a screencast). Since the speed of a simulation playback depends on the current load of both the CPU and GPU, recording a simulation playback via screencasting generally leads to

ACM ISBN 978-1-4503-5764-7/18/07...$15.00
https://doi.org/10.1145/3205651.3208292

[1] https://review.github.io/
[2] https://github.com/review/review.github.io

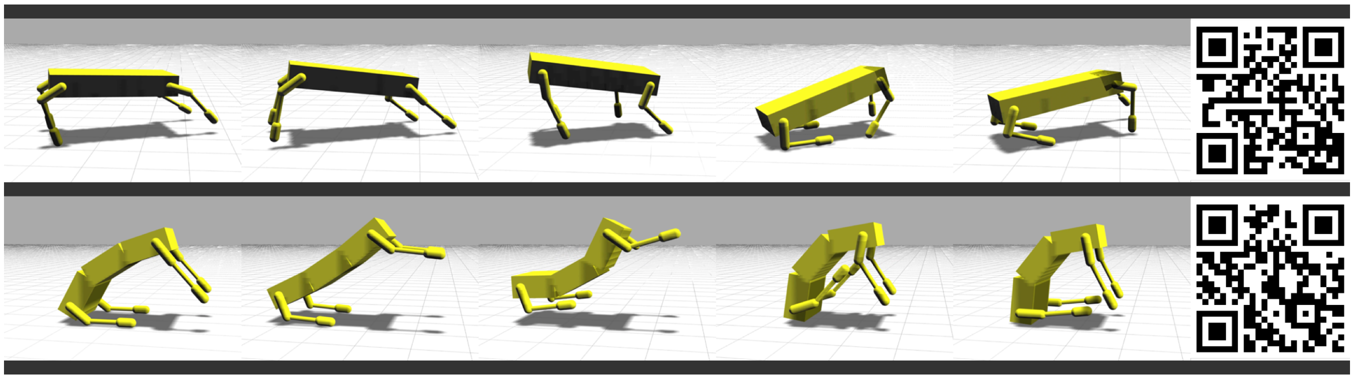

Figure 1: Evolved gaits for a quadrupedal animat: (top) galloping, (bottom) hopping. Interactive visualizations of the galloping and hopping gaits can be found, respectively, at the following links: http://bit.ly/2HYXS7u and http://bit.ly/2HVY9bc; or by visiting the links specified by the QR codes.

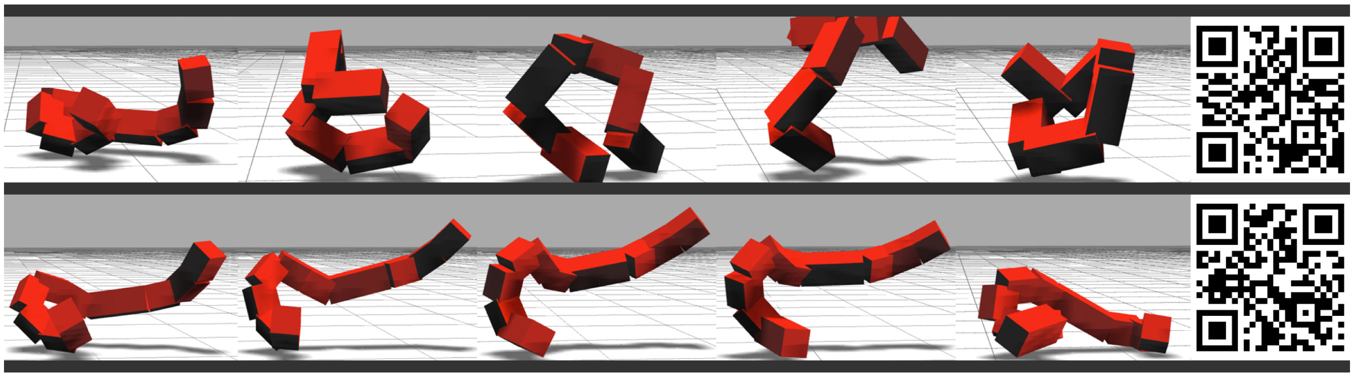

Figure 2: Evolved gaits for a worm-like animat (adapted with permission from Moore et al. [8]): (top) tumbling, (bottom) hopping. Interactive visualizations of the tumbling and hopping gaits can be found, respectively, at the following links: http://bit.ly/2vH4Knp and http://bit.ly/2HnHa4C; or by visiting the links specified by the QR codes.

videos with irregular frame-rates. Animation software, such as Review, does not have these drawbacks.

Review was originally developed to share visualizations among colleagues. Essentially, a tool was needed to solve two common questions in evolutionary robotics: (1) how can we share behavior results without needing to generate and email (or upload) large video files, and (2) how can we enable collaborators to manipulate the scene's camera as the visualization is playing. One method for achieving these two goals is to setup the same simulation and graphics environment on the machines of all collaborators, such that everyone is able to repeat the same experiment when given the same configuration (i.e., control and morphology parameters and environment initial conditions). To ensure repeatable results, in ER research this would require all collaborators to install the same physics and graphics libraries. This works for small teams and simple software packages, however, it becomes untenable when dealing with more complex systems.

Instead, Review has the ability to playback visualizations, inside a browser, when provided a log file. Thus, the workflow works as follows: (1) a researcher runs an evolutionary experiment (likely with the aid of a compute cluster), (2) interesting solutions are re-run, with visualization logging enabled (see section 2 for details), to generate log files, (3) log files are shared with other researchers (e.g., via email or uploaded to a website), and (4) log files are run with Review so that everyone can see the same results. Importantly, everyone using Review can independently adjust the playback speed, color of objects, and camera.

Review is meant to be complementary to both image sequences and videos. Many researchers will not have access to the Internet while reading a research paper, and for many other cases video will suffice. A web-based simulation viewer is meant for the following scenarios:

(1) sharing research with collaborators that need the ability to examine the scene from their own perspective, and
(2) sharing difficult-to-visualize behaviors in such a way that readers can manipulate the camera to better understand evolved behaviors.

2 REVIEW

Due to recent advances in web technologies (e.g., WebGL and HTML5), web-based visualizations have become increasingly prevalent [3]. Applications include: data visualization, education (e.g., viewing the human anatomy or the structure of a molecule), content creation (i.e., creating 3D models and other assets), gaming, and structure visualization (e.g., geospatial and architectural) [3, 9, 10]. The convenience of web-browsers and their increased performance has led to this increase in web-based applications.

2.1 Usage

Before we discuss Review in detail, we provide the following proposed usage for using Review as part of an ER study:

(1) Run evolutionary experiments
(2) Generate animation data (in Review log file format) for *interesting* results
(3) Load animations in Review
(4) Configure the scene (change materials and environment)
(5) Export animation file (Review log file and/or glTF formats)
(6) (optional) Generate high-quality videos using glTF
(7) Share animations with other researchers

The final step above can be achieved in several ways. First, Review log files can be directly sent to collaborators (e.g., via email). A log file saved on a local computer can be opened by Review through a file browser or by dropping the file onto the browser window. Alternatively, log files can be hosted on any server and automatically fetched by Review in the following manner: https://review.github.io/?log=https://raw.githubusercontent.com/review/review.github.io/master/examples/simple-sphere.json. Here, we have provided the URI of a log file to Review via an HTTP URI query component (note the "?" after the domain and the key-value pair that follows). When embedding links to an animation inside of a research article, it can often be desirable to shorten the URI with a URL shortener (such as Bitly[3] and Ow.ly[4]) . For example, this shortened link can be used in place of the previous link: http://bit.ly/2HOZQcY (this usage is also shown in the caption of Figures 1 and 2). Additionally, a QR code can be generated for the URL and paired with the image sequence as shown in Figures 1 and 2. This will enable readers to print a copy of an article and view the animation on their smart-phone using a QR code reader. Finally, Review-based animations can be embedded into any website using an HTML **iframe**. This allows users to see animations in a similar manner to an embedded video, but they will be able to manipulate the viewing angle, and change the color of objects in the scene.

[3]https://bitly.com/
[4]http://ow.ly/

2.2 Interface

A screen-shot of the Review interface is shown in Figure 3. Review provides two methods for loading a log file (the log files is described in the next section): (1) log files can be loaded from the local machine either by dragging the file onto the browser window, or by choosing a file from the open-file dialog box that is available when no log file is currently loaded (not shown), or (2) log files can be specified via an HTTP URI query component and loaded automatically by Review from remote servers (as demonstrated in the "Usage" section above).

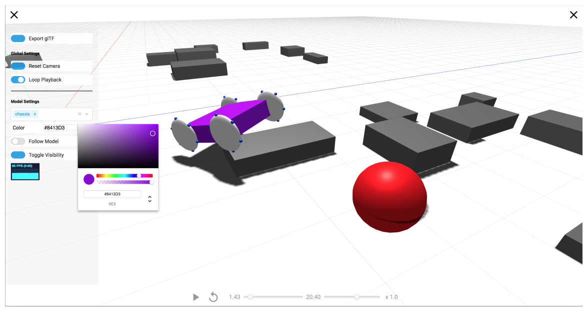

Figure 3: A screen-shot of Review with a UGV simulation in progress. The color palette can be used to change the color of each object in the scene.

This image depicts a scene of a transformable wheel UGV and several obstacles; the UGV's chassis color is currently being changed with a color picker that can be accessed in the settings pane. Review's settings pane includes several other expected features, for example, the ability to follow a specific object with the camera, reset the camera to its original settings, change the visibility of objects in the scene, and export the current animation. The settings pane can be hidden from view when it is unneeded. A standard set of playback controls are available at the bottom of the screen. One of the most useful controls is the progress bar, which can be *scrubbed* forward and backwards. Scrubbing refers to the ability to drag the progress icon to quickly navigate through time while watching the scene update on-the-fly. The playback speed can be adjusted from -5 to 5, where a negative number indicates that the scene will play in reverse.

2.3 Log File Format

One motivating factor for developing Review was the lack of an alternative application with a suitable file format. Most similar tools (described in section 3) require a complex file format that is difficult to generate alongside a physical simulation–likely because most 3D object viewers anticipate that visualization data is being generated by an authoring tool like Blender [1] or Autodesk Maya [11]. Common 3D file formats that include animations (e.g., COLLADA, FBX, glTF, and OpenGEX) focus on making it easy to load binary data and immediately send it through a GPU-centric graphics pipeline. For example, the following components are needed to specify a moving sphere in the glTF file format:

(1) A binary data buffer specifying the 3 dimensional positions of all points comprising the sphere, the surface normals of all points, and texture coordinates;
(2) A set of buffer accessors (called "views") for specifying how to extract positions, normals, and texture coordinates from the binary buffer;
(3) A material object that specifies various properties of the sphere's surface (i.e., roughness, metallic factor, and base color);
(4) A mesh object that groups together the buffer accessors associated with a single object and includes a reference to the sphere's material;
(5) A node object that refers to the above mesh and specifies the sphere's relationship to all other objects in the scene (even if no other objects are present);
(6) A scene graph specifying that the sphere is a root object (it's transform does not depend on any other objects);
(7) An animation object that specifies how the sphere is transformed (typically translations, rotation, and scale) over time.

glTF version 2.0 was released in 2017 and is a relatively simple specification compared to the other formats listed above. In contrast, the Review file format was specifically designed to be easily generated while running a simulation. An example of the Review log file format is provided in Listing 1. This is a visualization of a sphere that moves in a square pattern and pauses for one frame when it reaches its position defined in the fourth frame. This example animation can be viewed at the following link: http://bit.ly/2HOZQcY.

Listing 1: Review Log File Example

```
{
  "name": "Sphere Example",
  "timeStep": 0.25,
  "objects": [{
    "name": "sphere1",
    "mesh": "sphere"
  }],
  "frames": [
    { "sphere1": { "t": [0, 0, 0] } },
    { "sphere1": { "t": [1, 0, 0] } },
    { "sphere1": { "t": [1, 0, 1] } },
    { "sphere1": { "t": [0, 0, 1] } },
    {  },
    { "sphere1": { "t": [0, 0, 0] } }
  ]
}
```

The log file is a JSON file with a specific schema [12]. Here we describe a basic file, but the full schema can be found in the Review Git repository. This file format is human readable and writable, and nearly all programming languages include a library for manipulating JSON data. The Review logging library[5] used to generate the files linked in this study has been provided, and it contains less than 100 lines of C++ code and has only one external dependency (a C++ JSON library). This format has four required fields. The **name** (string) and **timeStep** (number) provide a unique name for the visualization and the time elapsed between frames, respectively.

objects (array) is a list of all objects that are present in the scene. In the example, there is only one object and only the required object fields are specified. Each object must have a unique **name** (string) and a **mesh** (string). The **mesh** value denotes either a primitive (*cube*, *cylinder*, or *sphere*) or a URI to an external resource (e.g., an STL or COLLADA file). In addition to these fields, several other attributes can be specified. The most prominent fields include: a physically based rendering (PBR) material, translation, rotation, and scale.

frames (array) is a list of all frames in the scene. The time lapsed between each pair of consecutive frames is always **timeStep**. Each frame is a JSON object where the keys (left side of the colon) denote objects named in the **objects** array, and the values (right side of the colon) denote object transformations. In the example, the only attribute of *sphere1* that changes is its translation. The most common animation attributes are translation (**t**) and rotation (**r**), but other types can be specified (e.g., scale). As exemplified by the fifth frame, this log file format allows for empty frames. An empty frame signifies that no objects are in motion during that frames time step. Allowing for empty frames greatly reduces the file size when many objects are in a given scene but only a few are in motion at a specific time. For example, for the UGV visualization in Figure 3 all 30 boxes are dynamic, but only one or two boxes are pushed by the UGV in any given frame. Although frame data may be omitted, currently the empty frames themselves must be present to ensure that the duration of the simulation is accurate–the log file format does not currently allow for variable **timeSteps** between different frames.

2.4 Technical Details

After Review loads a JSON-format log file, it converts it into glTF 2.0[6]. glTF is a transmission format specified in JSON that is used for loading and saving 3D models and scenes–an example of the contents of a glTF file are described above. It is advantageous to convert the Review log file into a format that is readily loaded by graphics engines. A custom graphics engine could be constructed around the Review log format (indeed, an earlier version of Review used a custom engine), however it is beneficial to use a full-featured graphics engine with state-of-the-art features and optimizations. We chose glTF due to its wide support and relatively simple specification. Thus, Review maintains the benefit of a simple file format while also gaining the capabilities of more capable rendering libraries.

[5]https://github.com/review/logger-cpp
[6]https://github.com/KhronosGroup/glTF

Review uses three.js [2] to render animated scenes to a browser window. three.js includes a modern rendering pipeline built on WebGL [5], great support for animations, and the ability to import glTF files. WebGL is a JavaScript API for rendering 3D graphics within a browser. WebGL enables GPU accelerated rendering in a similar fashion to OpenGL (specifically OpenGL ES). GPU shaders can be customized to produce stunning visuals that are rendered in the browser in real-time. WebGL is supported on many systems, including most modern desktop browsers and many browsers running on iOS and Android.

three.js is a cross-browser JavaScript library created to make 3D graphics applications. It includes many features for building complex scenes from simple components: different forms of lighting, different material and texture models, support for importing standard mesh formats, and an animation system. Perhaps the most important feature of the three.js animation system is *interpolation*, whereby three.js will interpolate transformation between frames. In the moving sphere example above, the time between frames is 0.25s, however, most computers will have no trouble rendering the scene at 60 frames-per-second, which means that three.js will use interpolation to generate roughly 15 frames in-between each frame. The impact of interpolation is most dramatic when the playback speed is reduced. For instance, when the playback speed is set to 0.25 three.js generates 60 interpolated frames, thereby providing a smooth animation. Without interpolation, animations would be choppy (similar to a slowed-down video).

3 RELATED WORK

Although web-based visualizers can be found for many different domains, there are only a few that have similar capabilities to those we describe here–namely, sharing animations with other collaborators and other researchers. Clara.io [4] is a cloud-based web-application for modeling, animating, and rendering scenes. Clara.io has an impressive list of features, including the ability to create models and scenes in the browser. With Clara.io you can send animation links to collaborators, but the data files must be hosted by Clara.io and the project is not open source. More importantly, Clara.io uses standard graphics formats, which makes it difficult to generate animation data as part of an evolutionary experiment work-flow. Sketchfab [15] is another alternative, and like Clara.io, Sketchfab hosts the data files and works with common graphics files. In contrast to Clara.io, Sketchfab focuses more on sharing and hosting graphics demos and less on authoring new assets and models. Verge3D [6] is an upcoming product that has similar features to both Clara.io and Sketchfab. Verge3D, however, focuses more on providing a means for creating 3D Web applications hosted by users. For example, it can be used to create an e-commerce website that delivers interactive 3D renderings of products.

Apart from these larger projects, Shen's Clay-Viewer [13] and McCurdy's glTF Viewer [7] have a similar interface to Review, and they load local animation files in glTF format. None of the above applications, however, provide a feature similar to our HTTP URI query component whereby an animation can be loaded by providing a link to the animation data in the URI. While these alternatives would enable sharing visualizations, they all require an extensive amount of work to generate visualization data (see the discussion on glTF in section 2). Another drawback of these websites is that many require a paid account to keep hosted files private, whereas with Review a researcher can choose to only share the log files with specific collaborators.

4 CONCLUSIONS

In this paper we have presented Review, a web-based tool for sharing evolutionary robotics visualizations with collaborators and other researchers. The interactive visualizations enabled by Review are complementary to current techniques such as providing image sequences and links to videos. Moreover, the simple log file format and export to glTF feature will enable researchers to import their simulation results into 3D author tools such as Maya, which will enable the production of high-quality videos more suitable for presentations. In the future, we anticipate adding the following features: (1) the ability to import more than one animation at a time so that researchers can compare the results from different trials in the same view, (2) support for additional mesh and material formats, and (3) the ability to generate videos by outputting frame data (which will provide a smoother video when compared to a screen recording).

ACKNOWLEDGMENTS

The authors would like to thank the contributions of Dr. Razib Iqbal, Michael Brattin, Kyle Finter, Garren Ijames, Brett Spatz, and Jesse Stewart from Missouri State University.

REFERENCES

[1] John M Blain. 2012. *The complete guide to Blender graphics: computer modeling and animation*. CRC Press.
[2] Ricardo Cabello et al. 2010. Three.js. Retrieved Apr. 3, 2018 from https://threejs.org/.
[3] Alun Evans, Marco Romeo, Arash Bahrehmand, Javi Agenjo, and Josep Blat. 2014. 3d graphics on the web: a survey. *Computers & graphics*, 41, 43–61.
[4] Inc. Exocortex Technologies. 2013. Clara.io. Retrieved Apr. 3, 2018 from https://clara.io/.
[5] Dean Jackson and Jeff Gilbert. 2011. WebGL. Retrieved Apr. 3, 2018 from https://www.khronos.org/webgl/.
[6] Yuri Kovelenov and Alex Kovelenov. 2018. Verge3D. Retrieved Apr. 3, 2018 from https://www.soft8soft.com/verge3d/.
[7] Don McCurdy. 2017. glTF Viewer. Retrieved Apr. 3, 2018 from https://gltf-viewer.donmccurdy.com/.
[8] Jared M. Moore, Anthony J. Clark, and Philip K. McKinley. 2017. Effect of animat complexity on the evolution of hierarchical control. In *Proceedings of the 2017 acm genetic and evolutionary computation conference*. Berlin, Germany, (July 2017). DOI: https://doi.org/10.1145/3071178.3071246.
[9] Jared M. Moore, Anthony J. Clark, and Philip K. McKinley. 2014. Evolutionary robotics on the web with webgl and javascript. In *Proceedings of the workshop on artificial life and the web 2014, held in conjunction with the fourteenth international conference on the synthesis and simulation of living systems (alife 14)*. New York, New York, USA, (July 2014). http://arxiv.org/abs/1406.3337.

[10] F. Mwalongo, M. Krone, G. Reina, and T. Ertl. 2016. State-of-the-art report in web-based visualization. *Computer graphics forum*, 35, 3, 553–575. DOI: 10.1111/cgf.12929. https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12929.

[11] Todd Palamar. 2015. *Mastering Autodesk Maya 2016: Autodesk official press*. John Wiley & Sons.

[12] JSON Schema. 2013. Json schema. Retrieved Apr. 3, 2018 from http://json-schema.org/.

[13] Yi Shen. 2018. Clay-Viewer. Retrieved Apr. 3, 2018 from https://pissang.github.io/clay-viewer/editor/.

[14] Karl Sims. 1994. Evolving virtual creatures. In *Proceedings of the 21st annual conference on computer graphics and interactive techniques*. ACM, 15–22.

[15] Sketchfab. 2013. Sketchfab. Retrieved Apr. 3, 2018 from https://sketchfab.com/.