Journal of Computer-Aided Molecular Design (2022) 36:677–686
https://doi.org/10.1007/s10822-022-00471-4

Prot2Prot: a deep learning model for rapid, photorealistic macromolecular visualization

Jacob D. Durrant[1]

Received: 13 July 2022 / Accepted: 1 August 2022 / Published online: 26 August 2022

Abstract

Molecular visualization is a cornerstone of structural biology, providing insights into the form and function of biomolecules that are difficult to achieve any other way. Scientific analysis, publication, education, and outreach often benefit from photorealistic molecular depictions rendered using advanced computer-graphics programs such as Maya, 3ds Max, and Blender. However, setting up molecular scenes in these programs is laborious even for expert users, and rendering often requires substantial time and computer resources. We have created a deep-learning model called Prot2Prot that quickly imitates photorealistic visualization styles, given a much simpler, easy-to-generate molecular representation. The resulting images are often indistinguishable from images rendered using industry-standard 3D graphics programs, but they can be created in a fraction of the time, even when running in a web browser. To the best of our knowledge, Prot2Prot is the first example of image-to-image translation applied to macromolecular visualization. Prot2Prot is available free of charge, released under the terms of the Apache License, Version 2.0. Users can access a Prot2Prot-powered web app without registration at http://durrantlab.com/prot2prot.

Keywords Prot2Prot · Molecular visualization · Web app · Machine learning · Proteins · Style transfer

Introduction

Molecular visualization is a critical structural-biology tool that provides valuable insights into the form and function of biomolecules. Artistically rendered photorealistic images can also inspire students and the public via education and outreach. Though well-known desktop programs such as VMD [1], UCSF Chimera [2], ChimeraX [3, 4], and PyMOL [5] were conceived principally as analysis tools, these programs can also produce striking images. But they understandably lack many of the advanced rendering techniques commonly used in the video game and film industries, and they are not designed to run in a web browser. Browser-based JavaScript molecular-visualization libraries such as Mol* [6], NGL Viewer [7, 8], 3Dmol.js [9], and Molmil [10] are also visually impressive, but they too are not designed to produce photorealistic renderings.

✉ Jacob D. Durrant
durrantj@pitt.edu

[1] Department of Biological Sciences, University of Pittsburgh, Pittsburgh, PA 15260, USA

Several desktop computer-graphics programs implement industry-standard rendering techniques, including Maya, 3ds Max, and Blender. Among these, Blender is notable because it is free and open source. However, none of these programs are designed for molecular visualization specifically. Several programs and plugins seek to address this shortcoming, including our BlendMol plugin [11], which allows users to easily import molecular models into the Blender environment. Though BlendMol greatly simplifies photorealistic molecular rendering, it still requires a good understanding of Blender, a program with a notoriously steep learning curve. And to produce high quality images and videos, even knowledgeable users must undertake the laborious process of setting up lighting, creating materials, positioning the camera, etc. Rendering itself is also computationally intensive, further limiting use.

We here describe a deep-learning model called Prot2Prot that imitates a Blender-rendered molecular image given a much simpler and easier-to-generate representation ("sketch") of a protein surface. Prot2Prot outputs an image that is often indistinguishable from a BlendMol-based visualization in a fraction of the time, allowing image "rendering" even in a web browser. Unlike the desktop tools

for molecular analysis described above [1–5], Prot2Prot is primarily a browser-based tool for generating photorealistic molecular renderings. Similarly, Prot2Prot is not meant to compete with desktop programs such as Blender/BlendMol [11], which are more difficult to use but provide exquisite control over lighting, materials, etc. Finally, Prot2Prot does not replace JavaScript libraries for browser-based molecular visualization [6–10], which display schematic (not photorealistic) renderings at high frame rates to enable smooth rotation and zooming. Rather, Prot2Prot aims to simply the process of generating photorealistic visualizations by abstracting away the complex settings typical of such a task while simultaneously offering the convenience of browser-based use.

The success of the Prot2Prot approach demonstrates how machine learning can serve as a valuable tool for enhancing scientific communication, with potential applications to fields beyond molecular visualization. We release Prot2Prot free of charge under the terms of the Apache License, Version 2.0. Users can access a Prot2Prot-powered web app without registration at http://durrantlab.com/prot2prot.

Materials and methods

Simplified protein-surface "sketches"

We first developed a simple 2D molecular representation that is easy to generate, even in slow and memory-limited environments such as web browsers (Fig. 1A). We represent each atom as a simple circle, sized according to the van der Waals radius and distance from the virtual "camera" (i.e., depth). Each circle is outlined in black to gray depending on its depth to emphasize atomic boundaries.

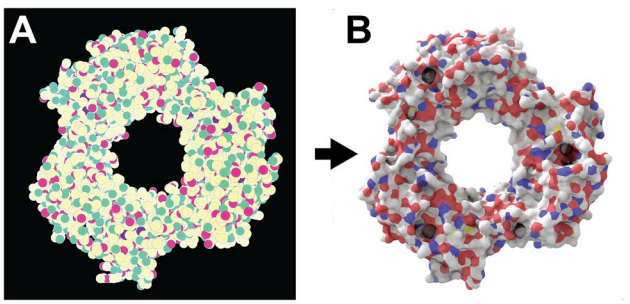

Fig. 1 Prot2Prot image mapping. Proliferating cell nuclear antigen (PCNA) bound to the PCNA-interacting motif (PIP box) of the DNA-dependent metalloprotease SPRTN (DVC1; 6099 atoms; PDB ID: 5IY4). A) The input image is a simplified 2D molecular representation that is straightforward to generate. B) The output image mimics the appearance of a photorealistic rendering of the same protein, as if created using Blender/BlendMol

The circles are colored according to carefully selected red, green, and blue (RGB) values. The red and green channels encode the atomic element. R/G values corresponding to carbon, nitrogen, oxygen, hydrogen, and phosphorus atoms are set at 100/100%, 100/0%, 0/100%, 0/50%, and 50/50%, respectively. All other atoms are encoded as carbons. Depth is encoded on the blue (B) channel. It is set at 100% for those atoms close to the camera and 0% for those atoms that are distant. A subtle three-step gradient is applied to each atom to capture its three-dimensional (spherical) shape.

To provide data sufficient for training a machine-learning model, we generated hundreds of representative protein-surface sketches (1024 × 1024). We first assembled a set of 49 diverse proteins from the Protein Data Bank [12] and added hydrogens to each of these proteins using *Reduce* [13]. To further augment the dataset, we used the 49 proteins to generate additional models. In some cases, we removed water molecules (if present) to generate new, water-free models. In other cases, we removed hydrogen atoms (important given that most models in the PDB lack hydrogens). When hydrogen atoms were retained, we randomly replaced occasional hydrogen atoms with rarer elements (e.g., sulfur, phosphorus, metals, halides). And we randomly rotated and scaled the models to capture proteins at many different angles and distances.

Blender/BlendMol-rendered molecular visualizations

For each protein-surface sketch, we rendered a matching photorealistic image (1024 × 1024) using Blender 3.0.0, an open-source computer-graphics toolset. We created custom Python scripts that load a PDB file into Blender using the BlendMol plugin [11]; automatically adjust the focal point of the Blender camera; create a fog-to-white effect; set the surface materials, lighting, and other parameters; and render a photorealistic image to disk using the Cycles path-tracing render engine.

Training a Prot2Prot model to map input sketches to rendered images

We trained Pix2Pix, a generative adversarial network (GAN) [14], to translate molecular sketches into the corresponding photorealistic protein images (Fig. 1; PyTorch Pix2Pix implementation available on GitHub [15]). In the context of this project, we call the model Prot2Prot rather than Pix2Pix. We used the default values for training, except we selected U-Net 128 as the generator architecture and used instance rather than batch normalization. For each of three photorealistic styles, we trained separate Prot2Prot models to generate 1024 × 1024, 512 × 512, and 256 × 256 output images,

respectively. To generate 512 × 512 and 256 × 256 images for training, we used ImageMagick [16] to scale the original 1024 × 1024 images. In all cases, we trained on roughly 1000 sketch/render pairs for 1000 epochs using the default initial learning rate, and then for another 1000 epochs as the learning rate decayed linearly to zero.

To further augment the data set available for training, we scaled the images by ~112% and then randomly cropped them at the original size (e.g., 256 × 256 images were scaled to 286 × 286 and then randomly cropped to produce 256 × 256 images). To allow the models to learn to mimic the consistent directional lighting of the rendered target images, we did not rotate or flip images, an otherwise common technique used for further data augmentation.

Because Prot2Prot is a GAN, there is no intuitive metric to monitor training progress or performance in an absolute sense (i.e., there is no straightforward loss function). Rather, a generator model learns to mimic real Blender-rendered images, while a discriminator model simultaneously learns to distinguish between the real and mimicked images. The performance of each model is always relative to the performance of its opponent. But visual inspection, albeit subjective, suggests the fully trained generator model mimics the actual Blender-rendered images surprisingly successfully.

Model conversion for use with tensorflow.js

We exported the trained PyTorch models to the ONNX format using the *torch.onnx.export* function. We then converted the ONNX files to the TensorFlow SavedModel format using the TensorFlow Backend for ONNX [17]. Finally, we converted the SavedModel files to the TensorFlow.js graph-model format using the *tensorflowjs_converter* command [18]. In this last step, we also applied 1-byte affine quantization to multiple nodes, which substantially reduced the file size without substantial impact on image quality.

Colorization

Prot2Prot produces images with consistent, predetermined color schemes. However, users can modify the color palette after inference, allowing some degree of customizability. A color-intensity matrix, *w*, determines how much a user-specified color influences various portions of the output image. The entries of the matrix range from 0.0 (no influence on the output image) to 1.0 (full influence).

The color-intensity matrix is calculated via element-wise multiplication of four image-derived matrices. First, to leave non-protein-surface regions unmodified, we convert the input "sketch" to a binary mask, where entries corresponding to the protein surface are set to 1.0, and other entries are set to 0.0 (Fig. 2A). Second, to influence well-lit protein-surface regions more than shadowed areas, we separately

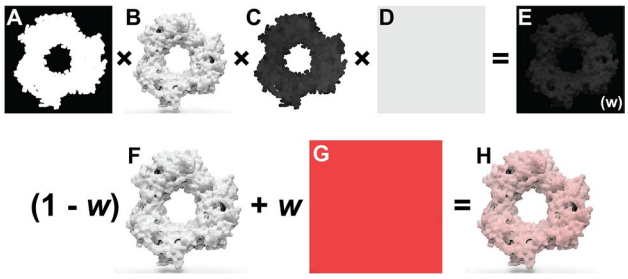

Fig. 2 Prot2Prot colorization procedure. **A** The mask matrix indicates which image regions include the protein surface. **B** The grayscale matrix distinguishes well-lit and in-shadow protein-surface regions. **C** The depth matrix indicates how far a protein region is from the virtual camera. **D** The color-strength matrix allows the user to further modify the strength of the colorization effect. **E** The final color-intensity matrix, called *w*, is calculated via element-wise multiplication of the four preceding matrices. **F** The original Prot2Prot output image. **G** A solid, user-specified color. **H** The final image, created by averaging the images in (**F**) and (**G**), weighting by the color-intensity matrix, *w*

convert the output "rendered" image to a grayscale matrix whose values correspond to averaged red, green, and blue values, scaled from 0.0 to 1.0 (Fig. 2B). Third, to preserve the fade-to-white fog effect, we create a third matrix from the blue channel of the input sketch image, which encodes depth (i.e., distance from the virtual camera). The values of this matrix range from 0.0 (most distant) to 1.0 (closest; Fig. 2C). Fourth, to allow the user to control the colorization effect's global strength, we create a matrix with identical entries equal to a user-defined color-strength parameter (Fig. 2D). After the element-wise multiplication of these four matrixes, the final matrix (Fig. 2E) can be optionally blurred to remove any sharp edges, per the user-defined color-blend parameter.

We use this color-intensity matrix to adjust the original Prot2Prot output image (Fig. 2F). A weighted average combines each pixel's red, green, and blue values with those of a solid, user-defined color (Fig. 2G). The pixel's color is unchanged if the corresponding color-intensity-matrix value is 0.0 and replaced by the user-defined color entirely if the corresponding value is 1.0 (Fig. 2H).

Browser implementation

We created a browser-based version of Prot2Prot following our established open-source approach [19–21]. The graphical user interface (GUI) is written in TypeScript using the Vue.js framework [22], the BootstrapVue CSS library [23], the TensorFlow.js machine-learning library [18], the Webpack module bundler [24], and Google's Closure Compiler [25].

The "Input PDB File" panel (Fig. 3A) allows users to load a PDB file into their web browser's memory by

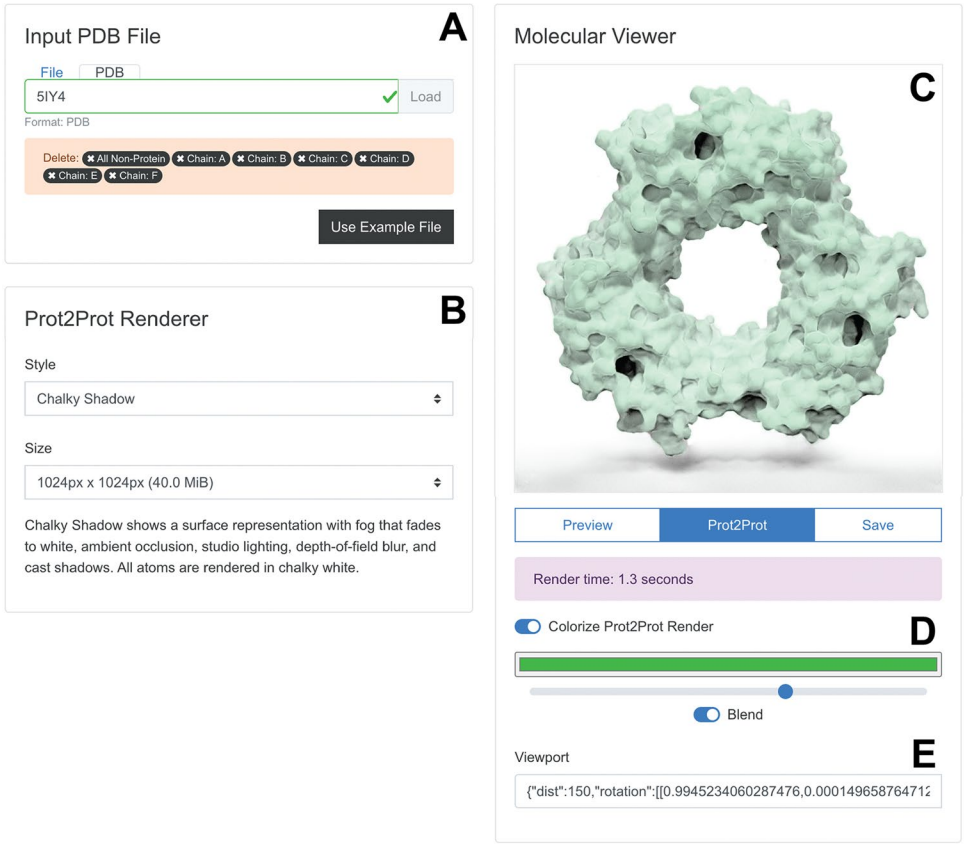

Fig. 3 Browser-app user interface. **A** The "Input PDB File" panel allows users to load and edit molecular structures. **B** The "Prot2Prot Renderer" panel allows users to specify the rendering style and image size. **C** The "Molecular Viewer" panel shows the rendered structure. **D** Colorize options allow the user to adjust the protein color. **E** The Viewport information can be copied and pasted to restore the rotation/zoom settings

selecting a file on their local computer or providing a PDB ID for remote download. Alternatively, clicking the "Use Example File" button automatically loads an example (PDB ID: 5IY4 [26]). Once the PDB file is loaded, the Prot2Prot user interface provides limited structure-editing options (e.g., users can remove ligands, water molecules, chains, etc.).

The "Prot2Prot Renderer" panel (Fig. 3B) allows users to choose from various rendering styles and image dimensions (see "Results and discussion" for a description). It also briefly explains the visual features of the selected style.

Users can position and display their molecules in the "Molecular Viewer" panel (Fig. 3C). Structures are initially shown in "Preview" mode as fields of atomic spheres that can be easily rotated and scaled using the mouse, mouse wheel, or touch gestures. Once ready, the user clicks the "Prot2Prot" button to generate the corresponding photorealistic image in the browser. The "Save" button allows users to save the viewer image. Users can also toggle the "Colorize Prot2Prot Render" setting to specify color, color-strength, and color-blending options (Fig. 3C, where green is selected). Finally, the app provides "Viewport" information that can be copied and pasted to restore the rotation/zoom settings (Fig. 3D).

Command-line-interface implementation

Aside from running Prot2Prot in a web browser, users can also access the model via a command-line interface (CLI) powered by the Node.js JavaScript runtime environment. CLI Prot2Prot is well-suited for rendering single images and image sequences, which can be combined into videos. CLI Prot2Prot provides several default animations, including "still," "rock," "turntable" (rotation about a user-specified axis), and "zoom." If a PDB file contains multiple frames, CLI Prot2Prot will also render protein dynamics, allowing users to visualize molecular dynamics simulations or interpolated protein structures (Online Resource 1).

Results and discussion

The Prot2Prot machine-learning model effectively renders photorealistic molecular representations via image-to-image translation of much simpler, easy-to-generate, molecular-surface "sketches." Prot2Prot illustrations are well suited for scientific publication, outreach, and education. CLI Prot2Prot can also generate animations of protein motions (Online Resource 1 and 2).

Description of rendering styles

We trained Prot2Prot models to mimic three distinctive rendering styles, which we call "Simple Surface," "Chalky," and "Chalky Shadow."

Simple surface

In the "Simple Surface" rendering style, carbon, oxygen, nitrogen, sulfur, and hydrogen atoms are light silver, red, blue, yellow, and white. Color support for other elements is limited. When rendering the photorealistic Blender images used for training, we applied two effects to give the final images a better sense of depth. First, we used Blender's mist pass to render more distant protein regions in lighter colors, producing a "fade-to-white" fog effect. Second, we used Blender's depth-of-field effect to focus the virtual camera on the protein surface directly in front of it, such that regions distant from that focal point appear slightly blurred or out of focus.

We also used several advanced lighting techniques to enhance photorealism. First, we applied a slight subsurface-scattering effect to all surfaces using Blender's Principled BSDF shader. When light hits many natural materials, it penetrates the surface and is scattered in the object's interior. After a light ray makes its way back to the surface, it leaves the object at a random angle, not the predictable angle typical of a perfectly reflective ("glossy") surface. Second, rather than light the scene with a single point or directional light, we used a public-domain, high dynamic range image (HDRI [27]) to surround and light the surfaces. High-dynamic-range (HDR) lighting prevents the darkest and lightest regions of the image from being saturated as perfectly black or white, allowing the viewer to see full detail across the entire image. Third, we applied ambient occlusion to the scene. This non-physical rendering technique approximates global illumination by darkening surfaces that are only partially accessible to the broader environment (e.g., enclosed pockets). After rendering the image using Blender's Cycles path-tracing render engine, we adjusted the color level using ImageMagick to ensure the background was precisely white, as typically required for publication-quality images.

We successfully trained our Prot2Prot models to mimic these Blender-rendered output images given a corresponding input "sketch image." When converted to the TensorFlow.js graph-model format, the final model takes up roughly 40 MB. Figure 4A, B show how the model has learned to mimic the fade-to-white-fog (*), depth-of-field (†), and ambient-occlusion (‡) effects of the Blender-rendered training images.

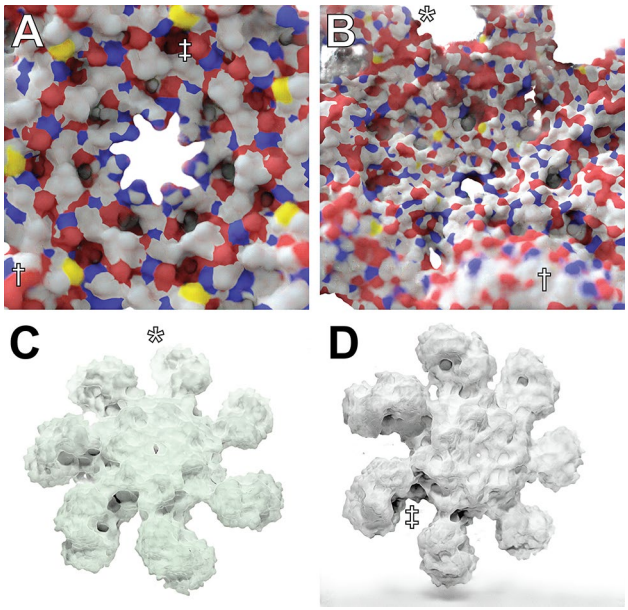

Fig. 4 An atomic resolution model of the human apoptosome obtained via electron microscopy (70,189 atoms; PDB ID 3J2T), visualized using Prot2Prot. **A**, **B** Simple Surface rendering style. **C** Chalky rendering style, colorized with a green tint. **D** Chalky Shadow rendering style. Examples of fade-to-white fog, depth of field, and ambient occlusion are marked with *, †, and ‡, respectively

Chalky

The "Chalky" rendering style also has fade-to-white fog, ambient occlusion, and depth-of-field blur. Unlike Simple Surface, Chalky shows all atoms in the same white material, without subsurface scattering. Instead, we set the "Roughness" and "Clearcoat Roughness" settings on the Principled BSDF shader to their maximum values to give the surface a highly diffuse appearance. Chalky uses a public-domain studio lighting setup obtained from blendswap.com [28] to light the proteins rather than an HDRI. After rendering the training images, we again adjusted the color levels using ImageMagick.

Trained Prot2Prot models successfully mimic these Blender-rendered output images as well. The Chalky models also take up ~40 MB, with similar run times in the browser. Figure 4C shows how Chalky images are particularly well suited to the custom colorization procedure (in this case, with a green tint) described in the Materials and Methods.

Chalky shadow

The "Chalky Shadow" rendering style is the same as the "Chalky" style, except the virtual studio lights are allowed to cast a shadow onto a pure-white floor below. The trained Prot2Prot models successfully mimic the shadows computed using advanced path tracing in Blender (Fig. 4D). Online

Resource 2 (bottom row) illustrates how these shadows even convincingly change according to the protein orientation. These models are also roughly 40 MB.

Video rendering via the command line interface

Command-line-interface (CLI) Prot2Prot also accepts multi-frame PDB files as input, allowing users to create animations of molecular dynamics simulations, conformational transitions, etc. Prot2Prot provides four different animation styles via its CLI (Online Resource 1). A "still" animation captures only the frame-by-frame motions of individual atoms without imparting any large-scale rotations to the entire protein. Alternatively, three whole-scene rotation animations can further facilitate visualization: "rock," "turn table," and "zoom."

To demonstrate these animation styles, we first used UCSF Chimera [2] to generate a multi-frame PDB file of *S. cerevisiae* hexokinase 2 (*Sc*Hxk2). Specifically, we used Chimera's "Morph Conformations" tool to capture the transition between open and closed *Sc*Hxk2 structures extracted from a recent molecular dynamics simulation [29]. We created video animations of this transition from image sequences of 48 Prot2Prot-rendered trajectory frames (Online Resource 1).

These animations convincingly capture the *Sc*Hxk2 open-to-close transition, but the protein surfaces appear to "flicker." This subtle artifact arises because Prot2Prot renders each frame without regard for adjacent frames (i.e., the resulting animations lack temporal coherence). To address this issue, we used Prot2Prot to re-render the *Sc*Hxk2 trajectory to only twelve images. We then used the Real-time Intermediate Flow Estimation (RIFE) 3.1 algorithm [30], as implemented in the Flowframes software package [31], to interpolate between these twelve images. The resulting animations capture the same open-to-close transition but without the flicker (Online Resource 2). We had similar success using the commercial frame interpolation algorithm implemented in Adobe After Effects.

Compatibility and run times

We have tested the Prot2Prot Web App on all major operating systems and web browsers (Table 1), including some mobile devices. The Prot2Prot model is memory intensive, and the web app will crash if run on a device with a less capable graphical processing unit (GPU). Where possible, the app detects any crash and asks the user to (1) select a smaller output-image size or (2) use the central processing unit (CPU) rather than the GPU. Rendering on the CPU is slower but also less memory restrained.

Prot2Prot currently runs fastest on Chromium-based browsers (e.g., Google Chrome, Microsoft Edge, etc.) because these browsers support OffscreenCanvas. On other browsers (e.g., Firefox), TensorFlow.js must use the CPU to run inference rather than the GPU. Users can already enable OffscreenCanvas in Firefox via the advanced configuration preferences, suggesting future versions will enable it by default.

We tested CLI Prot2Prot on Ubuntu Linux running Node.js 16.13.2. The Node.js runtime environment is

Table 1 Prot2Prot compatibility

Prot2Prot web app	
Operating system	Browser
macOS MONTEREY 12.1	Chrome 100.0.4896.30
macOS Monterey 12.1	Firefox 98.0
macOS Monterey 12.1	Safari 15.2
Microsoft Windows 10 Enterprise 10.0.19042	Chrome 99.0.4844.51
Microsoft Windows 10 Enterprise 10.0.19042	Edge 99.0.1150.39
Microsoft Windows 10 Enterprise 10.0.19042	Firefox 98.0
Ubuntu Linux 20.04.4 LTS	Chrome 99.0.4844.51
Ubuntu Linux 20.04.4 LTS	Firefox 98.0
Android 12	Chrome 99.0.4844.58
Android 12	Firefox 98.1.1
iOS 15.3.1	Safari 15
Command-line-interface (CLI) Prot2Prot	
Operating System	Node.js
Ubuntu 20.04.3 LTS	Node 16.13.2

We tested Prot2Prot on multiple operating systems, browsers, and Node.js versions

available on all major desktop operating systems, so we expect CLI Prot2Prot to be broadly compatible as well.

Aside from benefiting from broad compatibility, Prot2Prot also produces high-quality images much faster than dedicated 3D modeling programs such as Blender. Prot2Prot does not require users to set up lights, cameras, materials, etc.—setup activities that typically take much longer than rendering the image itself. But beyond eliminating the need for this laborious setup, Prot2Prot also has improved render times. To demonstrate, we rendered a test scene using Blender 3.2.0 on a MacBook Pro with an Apple X chip. The Blender Cycles path-tracing engine took roughly one minute to generate a 1024×1024 image using the GPU Compute device (Apple M1 Max GPU). In contrast, the Prot2Prot web app running on the same machine (Chrome browser) generated a similar image in only 1.2 s once the WebGL shaders had compiled (~6 s). Rendering times vary substantially depending on the available software and hardware (e.g., GPU vs. CPU). For example, older versions of Blender (e.g., 3.0.0) do not support GPU rendering on Apple hardware, and Prot2Prot does not run as quickly when using the CPU version of TensorFlow.js (as required, for example, in Firefox and Safari). But this comparison nevertheless demonstrates that Prot2Prot can dramatically accelerate photorealistic molecular visualization without requiring expertise in 3D modeling.

Visual comparison with other software packages

Figure 5 compares a Prot2Prot rendering to renderings produced by other popular molecular-visualization packages. Prot2Prot has learned advanced rendering techniques such as lighting and subsurface scattering, so users need not undertake the laborious process of setting these techniques up themselves. Rendering a Prot2Prot image is thus as simple as loading the protein, rotating and zooming, and pressing the "Prot2Prot" render button. In contrast, other molecular-visualization programs have many settings that users must adjust to modify the presentation. To normalize the effort invested in producing each image, we sought the path of least resistance when creating comparable renderings using other programs. We changed only those settings needed to set the protein representation to surface, to match atom coloring to the extent possible, and to set the background color to white. Figure 5A shows a Prot2Prot image rendered using the Simple Surface style. Figure 5B, D show renderings generated using the popular desktop molecular-visualization programs PyMOL [5], UCSF Chimera [2], and VMD [1], respectively. Figure 5E, F show renderings generated using two popular web-based visualization programs, Mol* [6] and 3Dmol.js [9].

Limitations

Prot2Prot is a powerful, easy-to-use tool for photorealistic protein rendering, but it has several notable limitations. First, it is generally useful only for rendering protein

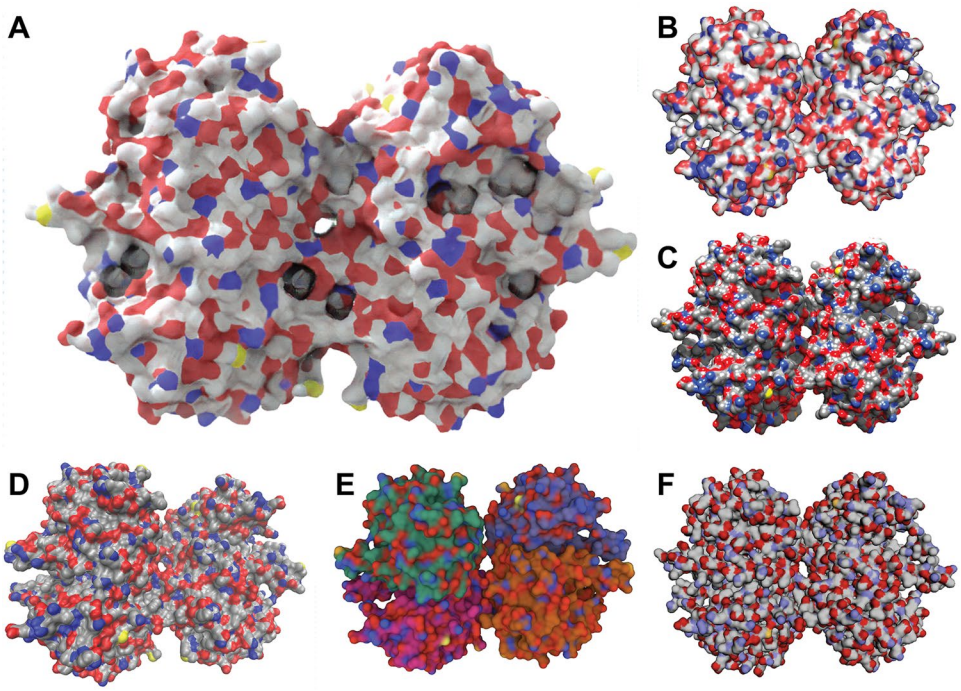

Fig. 5 Renderings produced by select molecular-visualization software packages. **A** Prot2Prot using the Simple Surface style. **B** PyMOL, a desktop program. **C** UCSF Chimera, a desktop program. **D** VMD, a desktop program. **E** Mol*, a web-based program. **F** 3Dmol.js, a web-based program. In all cases, we changed only those settings required to set the protein representation to surface, to match atom coloring to the extent possible, and to set the background color to white

surfaces. We attempted to train a Prot2Prot model to generate a cartoon-like image of protein tertiary structure given a sketch of the protein backbone atoms (Fig. 6A–C). Prot2Prot often correctly identified alpha helices and beta sheets, but misclassifications were frequent. Furthermore, it depicted alpha helices as elongated blobs rather than perfect cylinders.

The shadows rendered when using the Chalky Shadow style are generally impressive. Still, occasionally they appear to be more wavy than appropriate given the actual contours of the protein's profile (Fig. 6D, marked with †). Prot2Prot also sometimes renders a shadow "blob" in the lower-left-hand corner of its Chalky-Shadow output images (Fig. 6D, marked with ‡). Fortunately, image cropping can easily remove this small artifact.

Prot2Prot also often struggles to correctly render protein surfaces with positioning that differs substantially from that depicted in the training images. Artifacts typically occur when proteins are very close to the virtual camera (Fig. 6E, marked with *) or very distant (Fig. 6F). In the case of distant proteins, Prot2Prot appears to overemphasize the contribution of carbon atoms (Fig. 6F, colored in light silver). Finally, subtle checkered ("waffle") patterns occasionally appear when rendering proteins even at intermediate distances (Fig. 6G, marked with §). Rotating or scaling the molecule slightly generally eliminates these patterns.

Finally, Prot2Prot is trained to render protein surfaces, which are comprised primarily of carbon, oxygen, nitrogen, sulfur, and hydrogen atoms. The model is not trained to render macromolecules containing atoms of other elements (e.g., nucleic acids, which contain phosphorus). In practice, Prot2Prot can successfully render non-proteins when run using the Chalky and Chalky Shadow styles, which depend more on atomic positions that atom types. But running Prot2Prot using the Simple Surface style, which colors atoms by element, is sometimes problematic. Fortunately, in many cases the offending atom is obscured by other less problematic atoms (e.g., oxygen atoms, which often obscure an offending phosphorus).

Conclusion

The literature describes several other applications of image-to-image translation in medicine and biology. Examples include enhancing medical [32] and histopathological [33] images to facilitate diagnosis. Others have applied similar approaches to images obtained from electron [34] and fluorescence [35–37] microscopy with the goal of detecting gold nanoparticles or subcellular components. Still others have experimented with translating amorphous shapes to 3D representations [38]. But to the best of our knowledge, these

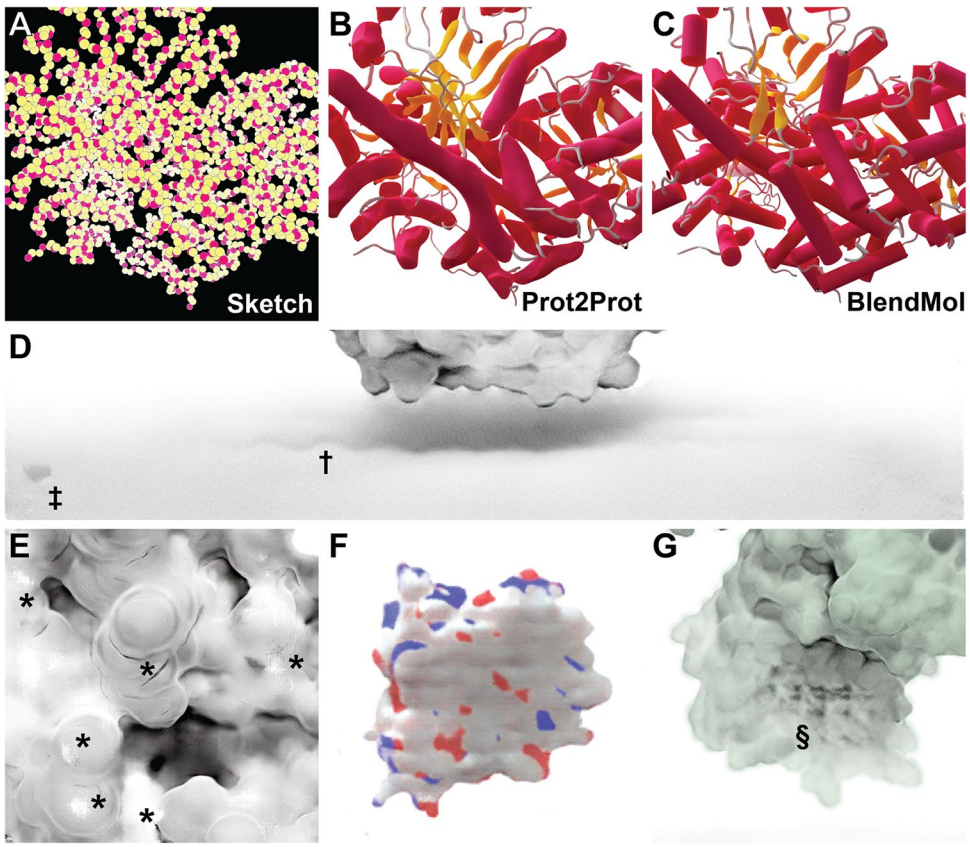

Fig. 6 Examples of Prot2Prot shortcomings. **A**–**C** Prot2Prot is best suited for rendering protein surfaces. It cannot accurately render a cartoon representation given a sketch of the protein backbone atoms. **D** The Chalky Shadow rendering style sometimes generates shadows that are excessively wavy (†). An artifactual shadow "blob" sometimes appears in the lower-left-hand corner (‡). **E** Viewing protein surfaces up close can produce artifacts (*). **F** Viewing protein surfaces at great distances tends to overrepresent carbon atoms (white). **G** On rare occasions protein surfaces may be subtly checkered even at intermediate distances (§)

approaches have never been applied to macromolecular visualization with the goal of producing photorealistic images for publication, outreach, and education.

Though the present work focuses on molecular visualization, it also suggests how machine learning algorithms can rapidly and effectively enhance scientific visualization generally. Blender specifically has been used to visualize many scientific phenomena, ranging from quantum wave functions [39] to tsunami hydrodynamics [40] to astrophysical data [41, 42]. A similar approach—generating simple representations of scientific data and converting those representations to higher-quality images—could be fruitfully applied in these other domains as well.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10822-022-00471-4.

Acknowledgements We thank the University of Pittsburgh's Center for Research Computing for computer resources and Harrison Green for help with the browser implementation.

Author contributions JDD is the sole author and is responsible for all aspects of the study design and execution. He also wrote the present manuscript describing the work.

Funding This work was supported by the National Institute of General Medical Sciences of the National Institutes of Health [R01GM132353 to J.D.D.]. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

Data availability The computer software and trained models that support the findings of this study are available free of charge, without registration, at http://durrantlab.com/prot2prot-download/ (Apache License, Version 2.0). Users can access the ready-to-use Prot2Prot-powered web app without registration at http://durrantlab.com/prot2prot.

Declarations

Conflict of interest The author declares that he has no conflict of interest. The author declares that he has no financial interests.

References

1. Humphrey W, Dalke A, Schulten K (1996) VMD: visual molecular dynamics. J Mol Graph 14(1):33–38
2. Pettersen EF, Goddard TD, Huang CC, Couch GS, Greenblatt DM, Meng EC, Ferrin TE (2004) UCSF chimera–a visualization system for exploratory research and analysis. J Comput Chem 25(13):1605–1612
3. Goddard TD, Huang CC, Meng EC, Pettersen EF, Couch GS, Morris JH, Ferrin TE (2018) UCSF ChimeraX: meeting modern challenges in visualization and analysis. Protein Sci 27(1):14–25
4. Pettersen EF, Goddard TD, Huang CC, Meng EC, Couch GS, Croll TI, Morris JH, Ferrin TE (2021) UCSF ChimeraX: structure visualization for researchers, educators, and developers. Protein Sci 30(1):70–82
5. Delano WL (2002) Pymol: An open-source molecular graphics tool. CCP4 Newslett Protein Crystallogr 40(1):82–92
6. Sehnal D, Bittrich S, Deshpande M, Svobodova R, Berka K, Bazgier V, Velankar S, Burley SK, Koca J, Rose AS (2021) Mol* viewer: modern web app for 3D visualization and analysis of large biomolecular structures. Nucleic Acids Res 49(W1):W431–W437
7. Rose AS, Bradley AR, Valasatava Y, Duarte JM, Prlic A, Rose PW (2018) NGL viewer: web-based molecular graphics for large complexes. Bioinformatics 34(21):3755–3758
8. Rose AS, Hildebrand PW (2015) NGL Viewer: a web application for molecular visualization. Nucleic Acids Res 43(W1):W576-579
9. Rego N, Koes D (2015) 3Dmol.js: molecular visualization with WebGL. Bioinformatics 31(8):1322–1324
10. Bekker GJ, Nakamura H, Kinjo AR (2016) Molmil: a molecular viewer for the PDB and beyond. J Cheminform 8(1):42
11. Durrant JD (2019) BlendMol: advanced macromolecular visualization in blender. Bioinformatics 35(13):2323–2325
12. Burley SK, Bhikadiya C, Bi C, Bittrich S, Chen L, Crichlow GV, Duarte JM, Dutta S, Fayazi M, Feng Z, Flatt JW, Ganesan SJ, Goodsell DS, Ghosh S, Kramer Green R, Guranovic V, Henry J, Hudson BP, Lawson CL, Liang Y, Lowe R, Peisach E, Persikova I, Piehl DW, Rose Y, Sali A, Segura J, Sekharan M, Shao C, Vallat B, Voigt M, Westbrook JD, Whetstone S, Young JY, Zardecki C (2022) RCSB protein data bank: celebrating 50 years of the PDB with new tools for understanding and visualizing biological macromolecules in 3D. Protein Sci 31(1):187–208
13. Word JM, Lovell SC, Richardson JS, Richardson DC (1999) Asparagine and glutamine: using hydrogen atom contacts in the choice of side-chain amide orientation. J Mol Biol 285(4):1735–1747
14. Isola P, Zhu J-Y, Zhou T, Efros AA (2017) Image-to-image translation with conditional adversarial networks. Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR). pp 1125–1134
15. Zhu J-Y (2022) Image-to-image translation in pytorch https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix. Accessed 11 Mar 2022
16. Team TID (2021) ImageMagick <https://imagemagick.org>. Accessed 5 July 2022
17. onnx/onnx-tensorflow (2022) Tensorflow backend for ONNX https://github.com/onnx/onnx-tensorflow. Accessed 11 Mar 2022
18. TensorFlow.js (2022) Machine learning for javascript developers https://www.tensorflow.org/js. Accessed 11 Mar 2022
19. Green H, Durrant JD (2021) DeepFrag: an open-source browser app for deep-learning lead optimization. J Chem Inf Model 61(6):2523–2529

20. Kochnev Y, Hellemann E, Cassidy KC, Durrant JD (2020) Webina: an open-source library and web app that runs AutoDock Vina entirely in the web browser. Bioinformatics 36(16):4513–4515
21. Young J, Garikipati N, Durrant JD (2022) BINANA 2: characterizing receptor/ligand interactions in python and javascript. J Chem Inf Model 62(4):753–760
22. You E (2022) Vue.js—the progressive javascript framework. https://vuejs.org/. Accessed 11 Mar 2022
23. BootstrapVue (2020) https://bootstrap-vue.org/. Accessed 11 Mar 2022
24. Koppers T (2022) Webpack. https://webpack.js.org/. Accessed 11 Mar 2022
25. Google (2022) Closure compiler: Google developers. https://developers.google.com/closure/compiler. Accessed 11 Mar 2022
26. Wang Y, Xu M, Jiang T (2016) Crystal structure of human PCNA in complex with the PIP box of DVC1. Biochem Biophys Res Commun 474(2):264–270
27. Majboroda S (2020) Photo Studio 01 HDRI. https://polyhaven.com/a/photo_studio_01. Accessed 11 Mar 2020
28. Blend Swap (2021) SomeDude (2021) Studio lighting setup. https://blendswap.com/blend/28426. Accessed 11 Mar 2022
29. Hellemann E, Walker JL, Lesko MA, Chandrashekarappa DG, Schmidt MC, O'Donnell AF, Durrant JD (2022) Novel mutation in hexokinase 2 confers resistance to 2-deoxyglucose by altering protein dynamics. PLoS Comput Biol 18(3):e1009929
30. Huang Z, Zhang T, Heng W, Shi B, Zhou S (2020) Rife: real-time intermediate flow estimation for video frame interpolation. arXiv preprint arXiv:201106294
31. N00MKRAD (2022) Flowframes—Fast Video Interpolation for any GPU. https://nmkd.itch.io/flowframes. Accessed 11 Mar 2022
32. Shin H-C, Ihsani A, Xu Z, Mandava S, Sreenivas ST, Forster C, Cha J (2020) Alzheimer's disease neuroimaging I. GANDALF: generative adversarial networks with discriminator-adaptive loss fine-tuning for Alzheimer's disease diagnosis from MRI. Springer, New York
33. Burlingame EA, Margolin AA, Gray JW, Chang YH (2018) SHIFT: speedy histopathological-to-immunofluorescent translation of whole slide images using conditional generative adversarial networks. Proc SPIE Int Soc Opt Eng 10581:29–35
34. Jerez D, Stuart E, Schmitt K, Guerrero-Given D, Christie JM, Hahn WE, Kamasawa N, Smirnov MS (2021) A deep learning approach to identifying immunogold particles in electron microscopy images. Sci Rep 11(1):7771
35. Shigene K, Hiasa Y, Otake Y, Soufi M, Janewanthanakul S, Nishimura T, Sato Y, Suetsugu S (2021) Translation of cellular protein localization using convolutional networks. Front Cell Develop Biol 9:635231
36. Lee HC, Cherng ST, Miotto R, Dudley JT (2019) Enhancing high-content imaging for studying microtubule networks at large-scale. PMLR, New York
37. Catchpole D, Shkeir N, Smith A (2020) Using generative adversarial networks to create multi-channel images of cells undergoing macropinocytosis
38. Horvath R (2019) Image-space metaballs using deep learning
39. Figueiras E, Olivieri D, Paredes A, Michinel H (2019) QMBlender: particle-based visualization of 3D quantum wave function dynamics. J Comput Sci 35:44–56
40. Giannakidis A, Giakoumidakis G, Mania K (2014) 3D photorealistic scientific visualization of tsunami waves and sea level rise. IEEE, pp. 167–172
41. Naiman JP (2016) AstroBlend: an astrophysical visualization package for blender. Astro Comput 15:50–60
42. Kent BR (2013) Visualizing astronomical data with blender. Publ Astron Soc Pac 125(928):731