Rethinking Data Visualization Software Architectures to Support Artistic Methodologies

Bridger Herman
herma582@umn.edu
Department of Computer Science and Engineering
Minneapolis, Minnesota

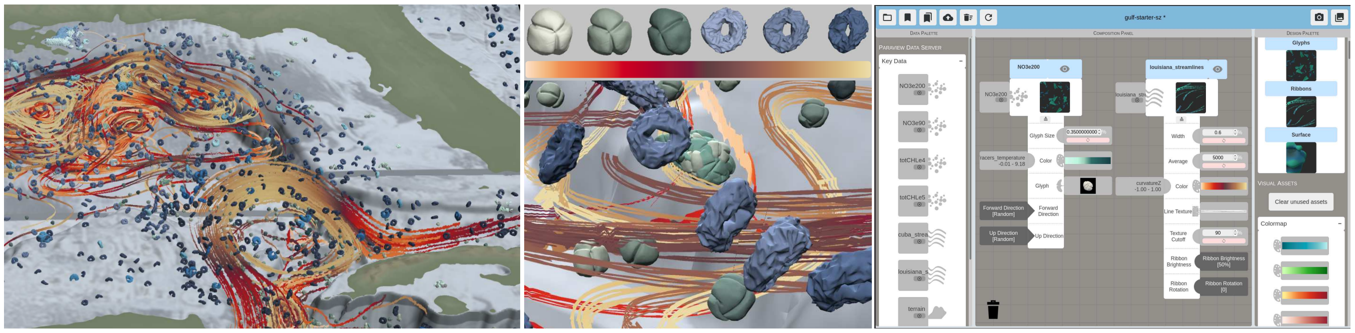

Figure 1: A visualization of biogeochemistry data in the Gulf of Mexico (left), a close-up view showing handcrafted elements of the visualization (center), and the visualization design interface used to make this visualization (right).

ABSTRACT

The challenges faced by scientists today require well-designed 3D multivariate data visualizations, and previous research has shown the importance of utilizing artists' expertise in color, line, shape, and form to help design these visualizations. However, visual design tools in current 3D visualization software often don't work well with artists' nonlinear creative methodologies due to the dominant influence of the data management utilities in that software. We present a three-step process to building a new visualization software architecture aimed at supporting creative processes: 1. Decoupling visual design tools from data management utilities, 2. Developing user interfaces for these visual design tools that support creative processes and connecting them to rendering engines that harness artists' media, and 3. Formalizing an artist-driven visualization state as a schema. We show the implementation details of the artist-focused visualization software architecture and an accompanying web-based user interface and discuss the implications that this type of architecture has on the future of collaborative 3D multivariate visualization.

CCS CONCEPTS

• **Applied computing** → **Arts and humanities**; • **Human-centered computing** → **Scientific visualization**; **Visualization systems and tools**; • **Computer systems organization** → *Distributed architectures*; *Client-server architectures.*

KEYWORDS

visualization software architecture, data visualization, user interfaces

ACM Reference Format:
Bridger Herman. 2021. Rethinking Data Visualization Software Architectures to Support Artistic Methodologies. In *Proceedings of (WPE).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Scientists and society as a whole have a serious need for artist-designed visualizations which have a broad scope of visual vocabulary that enchants and enlightens viewers. The necessity for artistic involvement in the visualization community has been substantiated by numerous accounts, such as Cox's work in visualizing high-dimensional spaces [7] and Samsel's application of sophisticated artist-designed colormaps to highlight crucial parts of a dataset [30]. Figure 1 (*left* and *center*) show a small sample of the visual variety that is produced with an artist-designed visualization of biogeochemistry data in the Gulf of Mexico, produced in collaboration with climate scientists and computer scientists. However, achieving a high level of collaboration and communication between technical and artistic fields has been difficult in practice due to disparate vocabulary and differences in the process for investigating new information in each field. Most traditional visualization software focuses on the technical utilities of visualization, including data management pipelines and processing algorithms. While these utilities are quite valuable, they also have a significant influence on the design of the visualization software architecture they are encompassed in. Based on first-hand comments from artists involved in data visualization, the visual design tools within these data-first architectures are often limited in the visual vocabulary they can express [29], and their user interfaces can introduce an additional cognitive load which interferes with the creative process. In this paper, we propose an alternative architecture which better supports artistic methodologies in the creation of complex visualizations.

With a more flexible visualization architecture both in user interface design and software design, it becomes feasible for seamless, collaborative workflows between scientists, artists, and technologists – the ultimate renaissance teams [8]. These collaborations are one of the driving factors behind new scientific insights and delivering understandable visualizations for the general public. In fact, many of the ideas in this paper stem from our group's interdisciplinary collaborative effort, the Sculpting Visualizations Collective (Sculpting Vis). The design and implementation of the new architecture is rooted in this philosophy that interdisciplinary collaborative teams have capabilities that far exceed those of the individuals on the team.

Our new, flexible architecture begins with decoupling and restructuring the components of modern visualization software, takes a tour through building new user interfaces to support creative processes in visualization, and follows up by connecting together the components of the new architecture. We also share lessons learned throughout the development process and future improvements to make the artist-focused architecture more stable and generalizable while retaining its flexibility.

2 RELATED WORK

2.1 Artistic Involvement in Data Visualization

Visualization has been a defining part of humanity's sense-making practices, ranging from ancient Mesopotamian tokens around 5500 B.C.E. [34], to Leonardo da Vinci's work in visualizations that fused art and science, to the present-day IEEE VIS conference where prolific visualization researchers come together to advance the possibilities of the field. Artists have been involved in data visualization for as long as the field has existed, and have lent their creativity towards better understanding of and accessibility to complex data. One common thread throughout artists' engagement with data has been a preference for physical materials and processes – now known as the field of *data physicalization*. Physicalizations are a diverse medium, ranging from palm-sized sculptures showing world data as 3D printed crystals [19], to park-sized installations at all-night art festivals centered on climate change [45], to data-driven ice sculptures representing glaciers that make viewers gasp as they melt [39]. Artistic explorations of data such as these physicalizations are one of the driving forces behind our work, because they have the capacity to engage an audience in ways that traditional, "sterile," computer-generated visualizations simply cannot [13]. Visual appeal and audience engagement are just two of the benefits of including artists in the data visualization design process.

Additionally, artistic involvement in data visualization has led to new scientific insights that wouldn't have been possible otherwise. Cox's work with visualizing supercomputer simulation data illustrates the utility of dataset-specific, artist-designed colormaps for highlighting important features in flow data [7]. Work by Samsel et al. also describes the importance of custom colormaps in environmental visualization applications [31, 32]. Artists intuitively follow principles and design guidelines based on their training, which mirrors practices established through other means (i.e. perceptual studies [5, 52]). Artists working with data visualizations provoke new imagery that has a handcrafted aesthetic [29]. Due to artists' training, these visualizations still follow best practices for achieving readability and contrast, leading to a "best of both worlds" between data and humanity [51].

In tandem with artists' engagement in visualization, there have been technical developments to support creativity in the vast digital world. One research area in Human-Computer Interaction that has emerged from the drive to support creative methodologies with technology is Creativity Support Tools, which strives to remove as many barriers as possible for artists engaging with technology [40], in essence "designing with low thresholds, high ceilings, and wide walls" [43]. Common barriers that exist in technology for creatives are rigid and inflexible user interfaces, and lack of access to prior examples of work [41]. Several solutions to these issues have been addressed, including use of metaphor and "fun" interfaces [42], visual exemplars [46], and sketch-based interfaces, which are an entire research area of their own. Creativity Support Tools in visualization aim to not only to provide user interfaces that work for the creative process but to actively promote the discovery of new visual combinations that work to show data in a new light.

2.2 Visualization Interfaces for Non-Programmers

Sketch-based interfaces have emerged as a key entry point for artists and other creative individuals to get involved in visualization design. Artists have previously used sketch-based interfaces to prototype 3D visualizations [16, 17], create custom illustrations of 2D fluid flows [35], sketch free-form glyphs [12, 36], and create multi-layered animated 2D visualizations of a variety of datasets [36]. In the current iteration of our software architecture, most sketching happens with traditional, physical media, but in the style of Buxton's "sketching user experiences," the visualization design interface in the architecture can also feel like sketching in its flexibility and affordance for rapid visual exploration [4].

One no longer needs direct access to a supercomputer and need not be an expert programmer to engage with data [50]; many tools now exist that harness the power of traditional programming languages for visual design while providing built-in commands that abstract common graphics code, making it easier for non-programmers to write "creative code" to achieve powerful visual effects. One such tool is Processing [11], which is based on the Java programming language. Many art installations have used Processing for data visualization and background graphics, including *Orbacles*, where a Processing-based flock simulation was projected on the ground [44] and *city flows*, which uses Processing to visualize bike sharing data from three cities [21].

Another approach to making computational visuals has been to avoid conventional programming languages and opt for an entirely visual experience for users instead. The Unreal game engine uses a node-based programming approach ("blueprints") which are in this style [47], and Blender's compositor and material editor use nodes to control the final look of a rendered piece of digital art [49]. Another style of visual programming with puzzle pieces has been frequently used as a learning tool for computational problem solving, for example with Blockly [23] and Scratch [27]. The visualization design interface in our architecture melds these two approaches, employing a puzzle-piece approach to building visualizations.

2.3 Visualization Systems

State-of-the-art visualization software systems have powerful data management utilities for scientists and technologists, but their user interfaces fall short of supporting artistic workflows well. This is not a coincidence – writing a piece of software that is up to "artist spec" is notoriously difficult [3], and for many visualization system developers, the artist spec is not a priority. We push back against this set of priorities because while the data processing utilities are a crucial part of the process, the end result is a *visual*ization, meaning the process for visual design should be given careful consideration in the structure of the software and its user interface. Giving this additional consideration becomes difficult because many modern visualization applications (i.e. ParaView [1] and VisIt [6]) are constructed as monolithic, self-contained applications. These applications do support nonlinear design processes to a certain degree, but to fully support artistic processes in software, modifications to their user interfaces are required to make them more flexible. The code for the user interfaces of these applications is technically modular and separable from the data management code, but in practice the UI toolkits these applications are built on are still too rigid to fully support creative methodologies. While there has been prior work emphasizing the "visual" in visualization software by building custom code for the visualizations and their design, these applications are limited in their generalizability between datasets. Some applications are built around a specific dataset (e.g. *city flows* [21]), and others are restricted to 2D data (e.g. Drawing with the Flow [35] and Visualization-by-Sketching[36]).

The Artifact-Based Rendering (ABR) technique [15] uses more generalizable approach at than previous work by using an integrated data pipeline from ParaView [1], a powerful, parallel-computing wrapper for the Visualization Toolkit (VTK) [37]. ABR supports artists in the creation of complex visualizations with 3D, multivariate, spatiotemporal data by allowing the use of handcrafted artifacts as elements in a visualization. Stage 1 of the ABR technique starts in an artist's studio with clay, ink, paper, and other forms of traditional media. During Stage 2 of the ABR technique, artists' creations (sculpture, ink washes, chine collé, etc.) are digitized using 2D and 3D scanning techniques, resulting in visualization assets (*VisAssets*) that are usable by the ABR visualization engine. Lastly, these VisAssets are used to encode variables a dataset, leading to visualizations with a decidedly handcrafted aesthetic [29]. The ABR technique forms the basis for the architecture presented in this paper, but it should be noted that there are marked differences between the "ABR Engine" mentioned in this paper and the "ABR Technique" showcased in the 2019 IEEE VIS paper. The most notable difference is that the visualization system used in the original paper still used a monolithic architecture; both the graphics and user interface for creating visualizations were tightly coupled to the data processing pipeline. As such, it still fell short of the "artist spec"; in practice requiring a computer scientist to "Wizard-of-Oz" the visualization controls while the artist made the design decisions.

Over the past three decades, there has been considerable work done in the world of distributed and parallel data visualization. Distributed visualization software follows the tradition of distributed systems in that each component of the system is programmed for accomplishing similar goals, and that messages are passed between components to fulfill these goals [48]. Due to the sheer size of the data involved in many of today's scientific simulations, supercomputers are often used for the data processing and visualization tasks associated with these data. Several solutions have been proposed to handle this, including distributing computation tasks of volume rendering [2], time-space partitioning [10], and remote control of visualization software running on a supercomputer [1]. While our architecture differs from these traditional distributed visualization systems in that its focus is the visualization *design* and not the data management, it is still a distributed system because there are several components communicating across a network to achieve a common goal.

3 DESIGN REQUIREMENTS

In visualization system building research, it is common practice to start with the data specification and move towards a final visualization. This approach elegantly succeeds for data processing tasks like creating isosurfaces, sampling glyphs from volumetric data, and seeding streamlines; however, it also leads to visualization architectures that are less inclined towards artists' creative methodologies. We take a different path. We bring the visualization design goals of our artist stakeholders to the forefront of the software development process, all the way from early mockups to the current implementation. Working closely with artists throughout the entire development process of the architecture allows us as software developers to focus on making the software accessible to humans while still retaining its entire technical capacity. In this section, we identify a number of essential design requirements for a visualization software architecture outlined by our artistic collaborators, the corresponding technical requirements this architecture must support, and a means of encoding these technical requirements using metaphor.

3.1 Artistic Requirements

3.1.1 Rapid, nonlinear exploration of visual design space. Artists working in their studios employ a nonlinear process which involves repeatedly making detailed changes in a piece of artwork and taking a step back to observe the work as a whole. This type of process encourages a creative mindset where artists can lose themselves in the work and instinctively rely on their training in visual principles. When applied to data, the process results in unique visualizations that can help stakeholders understand their data better [15] and are clearly influenced by each artist's individual style [29].

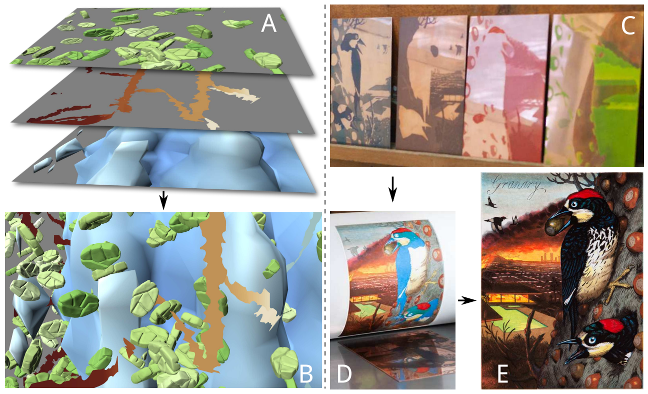

Figure 2: In computer graphics visualization, every visualization layer (A) is composited together into the final image (B). In intaglio printmaking, plates (C) are covered with colored ink and run through a printing press (D) to obtain impressions, many of which are combined into the final edition (E). *Right images Copyright 2020 Walton Ford and Wingate Studio; used with permission.*

3.1.2 Traditional-media artifacts. A big contribution to the individual style of artist-designed visualizations is the use of artifacts created with traditional artistic media. Such artifacts include hand-sculpted clay glyphs and colormaps based on other artwork (Figure 1 *center top*), as well as hand-drawn lines and textures (Figure 10 B and C). Incorporation of these artifacts is essential to fostering the creative processes of artists working with data [29].

3.2 Technical Requirements

3.2.1 Flexible user interfaces. Requirements 3.1.1 and 3.1.2 both necessitate a visualization design interface that is suited to creative processes. In the style of Creativity Support Tools [40], this means supporting artists in their exploration of the visual design space by enabling them to make rapid changes to the visualization and receive immediate visual feedback.

3.2.2 Real-time visualization control. To achieve the prior requirements, the visualization software architecture must support:

A. Quickly swapping visual elements and data
B. Modifying visual elements based on data
C. Viewing the visualization from various angles

3.3 Printmaking Metaphor

Effective visualization software for artistic stakeholders finds a balance to achieve all of the technical and artistic requirements, often through the use of metaphor; this is another essential requirement for the architecture. Intaglio printmaking was the metaphor upon which the architecture is built. Printmaking helps bridge the gap between the technical requirements of artist stakeholders and their technical implementations because it bears a resemblance to the way computer graphics are drawn and is a process that a majority of artists are familiar with. Figure 2 shows a step-by-step comparison of the printmaking process with the computer graphics rendering process.

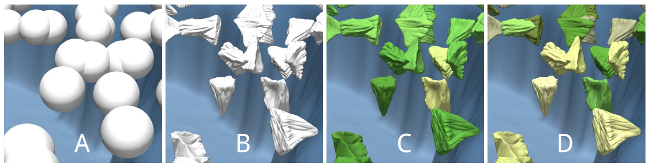

Figure 3: The construction of a glyph data impression, showing (A) applying the chlorophyll point sampling key data, (B) applying a glyph from the design palette, (C) applying a colormap and *Salinity* variable, and (D) applying the *Temperature* variable with the same colormap. Notice how the glyphs remain in the same location throughout because they all use the same Points key data object.

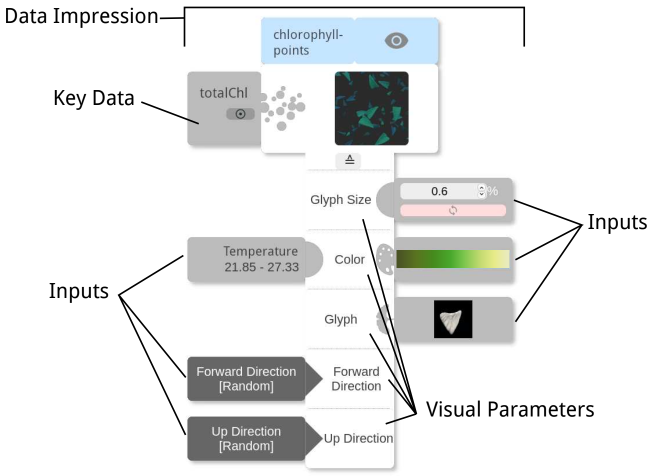

Figure 4: A Glyphs data impression (visualized in Figure 3D) with inputs populated by an artist.

4 IMPLEMENTATION

In this section, we present the design and implementation details of the new data visualization architecture that supports artist-focused visualization design interfaces. The entire software design process surrounding the architecture is focused on providing artists with a familiar means of creating a piece of visual art – in this case, a multivariate visualization. Thus, developing a visualization design user interface that fits well with artists' nonlinear creative processes was at the forefront of the team's minds as we created the architecture. To this end, we enumerated and deconstructed the elements of modern visualization software and put them back together into a new architecture that better supports artists' creative practice.

4.1 Terminology

The architecture borrows terminology from both intaglio printmaking and traditional data visualization. Terms from the ABR technique are also used, including VisAssets, the artist-created visual elements that can be used in a visualization (shown in Figure 1 *center top*). In printmaking, the plate with a registration (or the "structure") in printmaking is called a *key plate*, and each pass through the printing press is called an *impression*. These artistic terms are adapted for use in the data visualization world, hence we get *plates*, *data impressions* and *key data*. Key data are the geometric structures upon which we apply our designs to – *Points*, *Lines*, and *Surfaces*. One example of such a structure is the Points key data object with a glyph sampling of chlorophyll in the Gulf of Mexico shown in Figure 3, where glyphs are denser with higher chlorophyll concentrations and sparser with lower concentrations. Plates specify the available inputs on a particular type of data impression; this defines what type of key data they accept and other inputs like *Glyph Size* (numeric), *Color Variable* (scalar variable), and *Pattern* (VisAsset). Individual data impressions ("towers," shown in Figure 4) are initialized from plates, and are comprised of one or more key data objects along with all the visual designs applied to them. For instance, we might take the approach shown in Figure 3 and first apply a hue-varying green colormap, then apply a handcrafted triangular glyph to the chlorophyll point sampling.

The new metaphoric vocabulary is augmented with standard visualization vocabulary like *variable* and *dataset* to create a unified terminology in the architecture. We consider a dataset to be a collection of related key data objects, each containing variables. Variables may be present in more than one key data object. For instance, the *Temperature* scalar variable may be present in both the chlorophyll point key data and the ocean current streamlines key data, and the vector variable *FlowDirection* may be present in both of these key data as well.

4.2 Architecture Overview

In software systems, modularity is generally considered "good design", and has precedent in many systems today [38]. To achieve modularity and maximize the separation of the visual design tools from the data management and rendering utilities, the team embraced a distributed architecture where each component can run on its own computer and they communicate across a network. Decoupling the visual design tools from the data management utilities was the first step towards achieving a modular, distributed architecture. The architecture is separated into four main components:

- A web-based design user interface
- A web server
- A rendering engine
- A data host

Figure 5 depicts the components of this architecture and their relationships over a network. An artist creates the visualization in a web browser using the design UI (Figure 6), which is served from the web server on the graphics host computer. The server acts as an intermediary between the design UI clients and the rendering engine. A powerful graphics computer runs the rendering engine and displays the final visualization at interactive frame rates. The computer on which the data resides is known as the data host, which also supplies the data management utilities and sends preprocessed geometric data to the rendering engine. For the purposes of concrete demonstration, the examples shown in this paper use a combination of the Artifact-Based Rendering technique [15], ParaView [1], and a Python Django server to implement the components of the architecture, but each of these components may be swapped out at any time.

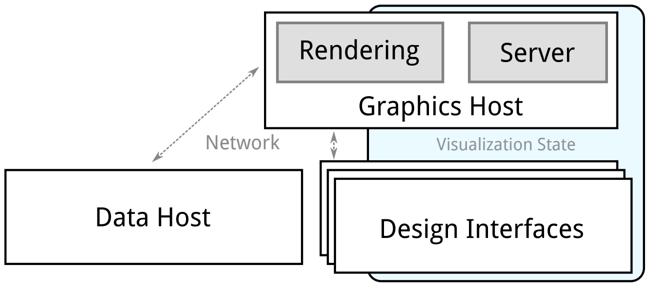

Figure 5: Data sources hosted on a supercomputer, data visualization graphics and a web server hosted on a powerful graphics computer, and visualization design interface clients all work together to allow artists to build visualizations of 3D multivariate data which are rendered at real-time frame rates. Visualization state is shared between the rendering engine, the server, and the design user interfaces.

4.2.1 Rendering engine. The ABR engine is the rendering engine in this implementation of the architecture, which uses the Unity game engine, C# scripts, and ShaderLab shaders to render the artist-driven visualization graphics. The engine supports creative methodologies in visualization by allowing artists to use their own work, often created in physical media, and is connected with a web-based visualization design user interface (Figure 6). The rendering engine accepts commands in real-time from the visualization design interface, supporting Requirement 3.2.2A. As artists construct a visualization using the interface, states can be saved to JSON files at the artist's discretion so they can come back to work on it later or share the state with others. If the rendering engine is running on a computer with a head-mounted Virtual Reality display connected, the resulting visualization can also be displayed in VR at interactive frame rates. The rendering engine also supports Requirement 3.2.2C by supplying standard 3D navigation controls in the visualization display window.

4.2.2 Data host. To obtain the data, the rendering engine connects through sockets to the data host. In the data host, volumetric data are preprocessed into geometric forms (points, lines, and surfaces), then further processed in the rendering engine to get the renderable meshes. Glyphs are passed through an instance-rendering shader, lines are rendered as a quad strip, and surfaces are rendered as a series of triangles.

4.2.3 Web server. On the user-facing side of the architecture is the web server which handles all connections and messaging with the design user interface. The server is run within the same file system as the rendering engine so that data and artist-created VisAssets can be shared between these components. For instance, the design UI fetches the colormap thumbnails directly from the server, and the rendering engine uses colormap files in the same directory to apply to the visualization. The server also passes messages between the design UI and the engine using WebSockets [9]. All messages between the design UI and the ABR engine are in JSON format.

4.2.4 Visualization design interface. The visualization design interface is built for web browsers in order to support as many devices

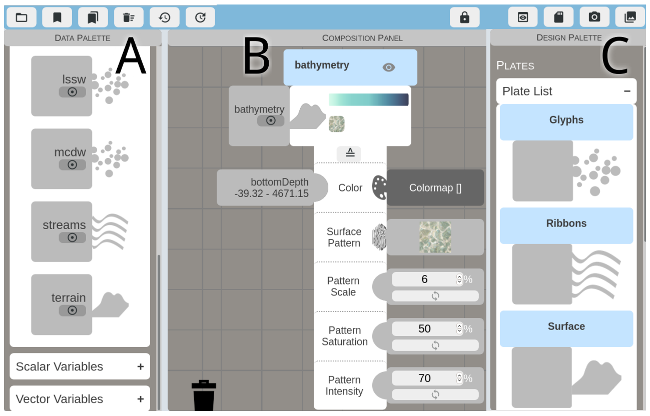

Figure 6: The design user interface, showing the data palette (A), the composition panel (B) with one data impression, and the design palette (C).

as possible, and to take advantage of the breadth of modern web libraries available. Specifically, the interface uses jQuery UI for most of its primary interactions, including dragging and dropping puzzle pieces. The interface also includes a colormap editing utility (shown in Figure 8) that takes advantage of the D3 visualization library [54] for showing the histogram of the variable attached to a particular colormap. Because it is implemented as a web app, the design interface can be run on a different computer from the visualization – even on a tablet across the country.

4.3 The Visualization Design Interface

Figure 6 shows the web-based 3-panel design interface that allows for rapid iteration on the visualization contents. When considering artists' need for fast, nonlinear iteration of visual designs (Requirement 3.1.1), the fundamental processes used in the construction of a visualization need to change from the traditional data-first approach. For instance, an artist may consider both the *Ribbons* and *Clay Forms* in Figure 7 to be "lines," and features like seamlessly changing from ribbons to clay forms is something most artists would expect from the software they work with. However, from a data-first computer graphics standpoint, rendering a ribbon (quad strip) is significantly different than rendering a series of instanced glyphs; in fact the easiest approach to make this kind of change is usually to make a new glyph layer, copy all the visual encodings from the old ribbon layer, and then discard it. The visualization design interface uses the metaphor of a "data impression" to convey the technical constraints of the computer graphics algorithms in a way that is consistent with artists' studio processes.

4.3.1 Data Impressions. Data impressions are the means with which artists create visualizations in the design interface. Each data impression is a single "layer" of the visualization, and artists build their visualization composition by creating many data impressions with different key data. Data impressions are initialized by dragging and dropping a plate from the design palette panel on the right into the composition panel in the center of the design user interface (see Figure 6). Figure 4 depicts a single Glyphs data impression which uses the chlorophyll point sampling from Figure 3 (D) and two artist-chosen VisAssets – a colormap and a glyph.

Data impressions have one or more key data. In Figure 4, the Glyphs data impression has a Points key data object attached to it. Like every variable and VisAsset in the interface, the key data puzzle pieces have evocative iconography showing an artist's interpretation of that type of key data. Key data are selected from the left data palette panel and dragged into matching data impressions in the center composition panel of the design interface (see Figure 6).

Additionally, data impressions each have a set of visual parameters that the artist can change. Each parameter may have one or more inputs, for instance in Figure 4 the *Color* parameter has two inputs: *Colormap* for the artist-created colormap VisAsset, and *Color Variable* for the scalar data variable to be encoded with that colormap. Similar to key data, scalar and vector variable puzzle pieces are dragged and dropped from the left data palette, and VisAssets are dragged and dropped from the right visual design palette into matching puzzle slots on a data impression in the center composition panel. Two methods are used to indicate puzzle piece compatibility – the iconography of the puzzle piece shape must match its input slot, and a subtle blue glow appears over each compatible slot when dragging a puzzle piece.

4.3.2 Building Visualizations. Using the interface, artists create visualizations in a manner similar to the processes they use in the studio. Data impressions provide a logical means of organizing components of a visualization, and inputs to these data impressions allow for quick, iterative changes to be made to a visualization at the drop of a puzzle piece (which addresses Requirement 3.2.1). We provide two concrete examples of working with the interface which showcase possible ways an artist may desire to create a visual outcome. Both examples use data from a Gulf of Mexico biogeochemistry dataset [25], and have Surface data impressions of the ocean floor and land for context. The generic process for getting started with a first data impression in the design user interface is as follows:

(1) Initialize a data impression from a plate
(2) Assign a key data object
(3) Assign VisAsset and variable inputs
(4) Iteratively adjust visual parameters

First, let us return to the previously mentioned situation where an artist wants to change the type of line for the ocean current streamlines south of Louisiana shown in Figure 7. Let's say that an artist wants to begin with representing these currents with Ribbons. A Ribbons data impression is instantiated, the proper key data are applied, and the artist chooses VisAssets for the *Line Texture* and *Color* visual parameters. At this point, the artist may decide that they want to use clay forms instead of ribbons to visualize the ocean currents. Recall that the computer graphics algorithm for rendering clay forms is entirely different from that of the ribbons. This constraint is directly encoded within the design interface: the artist cannot drag the Points key data of the line onto the Ribbons data impression – they must create a new Glyphs data impression and copy across each data variable and visual element to the new impression.

The second example (Figure 3) shows a point sampling of chlorophyll density near the western coast of Florida. A Glyphs data

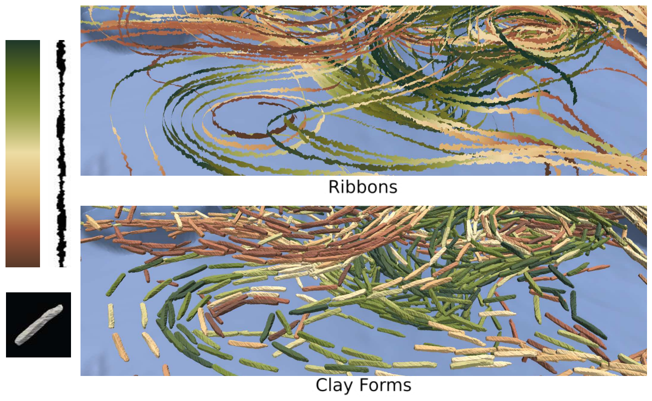

Figure 7: Two line impressions, one made with a ribbons plate and another made with a glyphs plate.

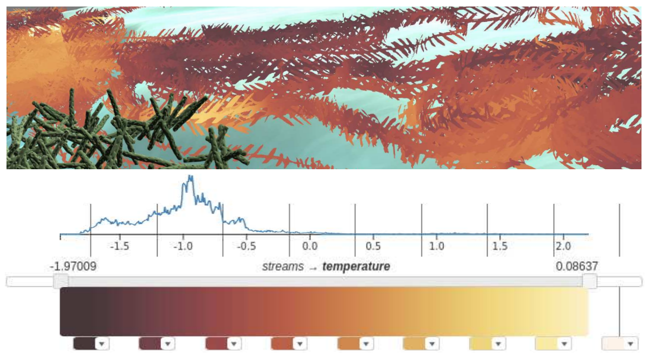

Figure 8: The colormap editor integrated with the visualization design user interface allows for on-the-fly changes to the colormap based on a histogram of real data values.

impression is instantiated, and the chlorophyll point sampling key data object is applied to the impression. Next, a glyph VisAsset is chosen from the design palette and applied to the *Glyph* input, a colormap from the design palette is then applied to the *Colormap* input, and a scalar variable *Salinity* is applied to the *Color Variable* input. Lastly, if the artist decides to visualize *Temperature* instead of *Salinity*, they need only drag in the *Temperature* scalar variable from the data palette to replace the *Color Variable* input.

4.3.3 Working with VisAssets. Artists to want to use their own work when creating visualizations, so any artist-focused interface for visualization design must keep this in mind (Requirement 3.1.2). The Artifact-Based Rendering technique allows artists to incorporate their own textures, colormaps, lines, and glyphs which affords the resulting visualizations a much greater visual vocabulary [29]. The design interface allows for two complementary strategies for working with VisAssets: downloading them from a curated library [28], and working with local copies.

Handcrafted artifacts (glyphs, colormaps, lines, and textures) are curated by artists in the Sculpting Vis Collective, and a series of ABR web applets [15] are used to convert them into usable VisAssets. In order to take full advantage of ABR, artists will often need to modify these VisAssets while designing a visualization. The design user interface features a colormap editor (Figure 8) inspired by ColorMoves [31] which allows the artist to create custom, dataset-specific colormaps and view the resulting color changes on the data in real-time.

4.4 Proposed Visualization Schema

We propose an additional component to augment the architecture presented so far – a visualization state specification based on a JSON Schema [26]. Schemas have a long-standing precedent for use in defining a visualization state; Snap-Together Visualization [22] uses a relational database schema and Vega-Lite [33] uses a JSON schema to define interactive visualizations. The schema describes a formalization of an artist-created visualization state, such as that shown in Listing 1 and Figure 9. The following describes the roles of this schema and their impact on the flexibility and stability of the architecture.

```
{ "impressions": [ {
      "uuid": "12598986-819c",
      "plateType": "Surfaces",
      "name": "Bathymetry",
      "inputValues": {
        "Key Data": {
          "inputType": "PointsKeyData",
          "parameterName": "Key Data",
          "inputGenre": "KeyData",
          "inputValue": "/Ronne/key-data/bathymetry" },
        "Color Variable": {
          "inputType": "ScalarVariable",
          "parameterName": "Color",
          "inputGenre": "Variable",
          "inputValue": "/Ronne/scalar-vars/depth" },
        "Colormap": {
          "inputType": "ColormapVisAsset",
          "parameterName": "Color",
          "inputGenre": "VisAsset",
          "inputValue": "f31cfdb3-0f2f" },
        "Pattern": {
          "inputType": "PatternVisAsset",
          "parameterName": "Pattern",
          "inputGenre": "VisAsset",
          "inputValue": "285a2582-48f6" } }
    }, {
      "uuid": "99e2b165-2e12",
      "plateType": "Glyphs",
      "label": "CDW",
      "inputValues": {
        "Key Data": {
          "inputType": "PointsKeyData",
          "parameterName": "Key Data",
          "inputGenre": "KeyData",
          "inputValue": "/Ronne/key-data/cdw" },
        "Color Variable": {
          "inputType": "ScalarVariable",
          "parameterName": "Color",
          "inputGenre": "Variable",
          "inputValue": "/Ronne/scalar-vars/temp" },
        "Colormap": {
          "inputType": "ColormapVisAsset",
          "parameterName": "Color",
          "inputGenre": "VisAsset",
          "inputValue": "f52dabab-5b11" },
        "Glyph": {
          "inputType": "GlyphVisAsset",
          "parameterName": "Glyph",
          "inputGenre": "VisAsset",
          "inputValue": "1af03108-f1ed" } } } ] }
```

Listing 1: Data impressions of visualization state depicted by Figure 9. *UUIDs have been shortened to fit on the page. The JSON schema that this state fragment conforms to may be found in the supplemental material.*

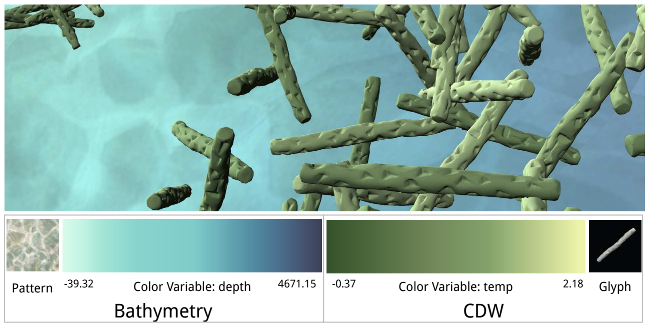

Figure 9: ABR visualization created by artist Francesca Samsel with a Glyphs data impression showing Circumpolar Deep Water (CDW) and a Surfaces data impression showing the Bathymetry below the Ronne-Filchner ice shelf, described by the JSON state in Listing 1.

4.4.1 Schema as a description of capabilities. The schema defines the plates that are available in the architecture. This allows developers to make algorithmic improvements to the rendering engine without affecting code in the user interfaces or destroying backwards compatibility with visualization states. The schema also provides definitions to every input so that the rendering algorithms can be improved or swapped out with complete transparency to the artist.

4.4.2 Schema as a contract. The schema also ensures that the visualization state representation is consistent across the architecture. The visualization state schema has four main components: data impressions, local VisAssets, scene information, and UI data. The most important part of the visualization state, data impressions, is a list of artist-created layers. Each data impression has a unique identifier, and its inputs are populated with the VisAsset identifiers, data paths, or primitives that the user has dragged and dropped in the artist-facing design interface. Other scene data such as lighting information is also initialized and adjusted through the interface. In addition, the state also stores VisAssets the artist has modified, as well as any UI-specific data that may need to be saved for a particular interface, such as the location of the data impression towers in the composition panel.

5 RESULTS

The architecture enables new remote multivariate 3D visualization collaboration in a way that was not previously feasible. Another important result for the architecture was the feedback and insights on the interface and visualization design provided by artists who used the interface. In this section, we describe the remote design sessions that were conducted with a variety of geographically distributed collaborators, as well as several insights from the artist stakeholders who used the interface.

5.1 Design Sessions Enabled by the Architecture

The network-based implementation of the architecture allowed our team to conduct design sessions with people across broad distances. During each design session, the participants used the web-based visualization design interface to build visualizations of curated datasets such as biogeochemistry in the Gulf of Mexico [53] and ocean currents under the Ronne-Filchner ice shelf of the Antarctic [24]. As necessitated by the ongoing pandemic, all design sessions were run entirely remotely via the networking and web capabilities of the architecture in conjunction with video conferencing software.

In the architecture's infancy, we facilitated two remote design sessions with artists. Both artists watched a short training video on the use of the interface prior to their sessions, and could ask as many questions as they wanted while creating a visualization. The first session was conducted with artist Deborra Stewart-Pettengill, who works with many forms of traditional artistic media and had no prior experience working with 3D scientific data. Stewart-Pettengill has lately been working with chine collé, a printmaking technique that involves thin pieces of colorful paper which are cut out and applied to the print when it is run through the press. The interaction with the visualization design interface allowed Stewart-Pettengill to use her chine collé work in a visualization of biogeochemistry data in the Gulf of Mexico (see Figure 10 B and C). During the session, Stewart-Pettengill iterated through a total of 12 glyphs, 7 lines, 8 textures, and 26 colormaps from the Sculpting Vis library. Three of the lines were processed into VisAssets from Stewart-Pettengill's chine-collé work, which took an ABR expert approximately thirty minutes total to accomplish before the session started. Additionally, all 8 textures were sourced directly from the chine-collé work, which took an ABR expert about 10 minutes total to convert into VisAssets. Session one took approximately an hour and a half total.

A second design session was held with artist Stephanie Zeller, who has worked with 3D scientific visualizations previously including custom colormap creation. Through the use of the Sculpting Vis web applets [28] and the colormap editor in the design user interface, Zeller created several new colormaps as well as making dataset-specific modifications to colormaps from the Sculpting Vis library to benefit the clarity of the variable they are applied to. During the session, Zeller used 14 glyphs, 9 lines, and 9 textures, and 56 colormaps. She edited 18 of these colormaps during the session using the colormap editor, and she created two colormaps from scratch using the Sculpting Vis web applets [28]. See Figure 1 (*left* and *center*) for a view of Zeller's visualization, featuring her custom colormaps in combination with other VisAssets from the Sculpting Vis library. Session two was conducted in two parts, the first about one hour long and the second nearly two hours.

A few weeks later after several improvements were made to the architecture and user interface based on feedback from the earlier artist sessions, another design session was conducted with 7 members of the Sculpting Vis Collective. This session was focused on the remote collaboration possibilities enabled by Johnson's approach to remote graphics rendering [14]. Each member of the design session viewed the visualization remotely through a VR headset or a desktop viewer if they didn't have access to a head-mounted display. During the session, the team was able to produce insights about the 3D multivariate data we were working with while one of the artists on our team worked rapidly on the visualization using the design interface.

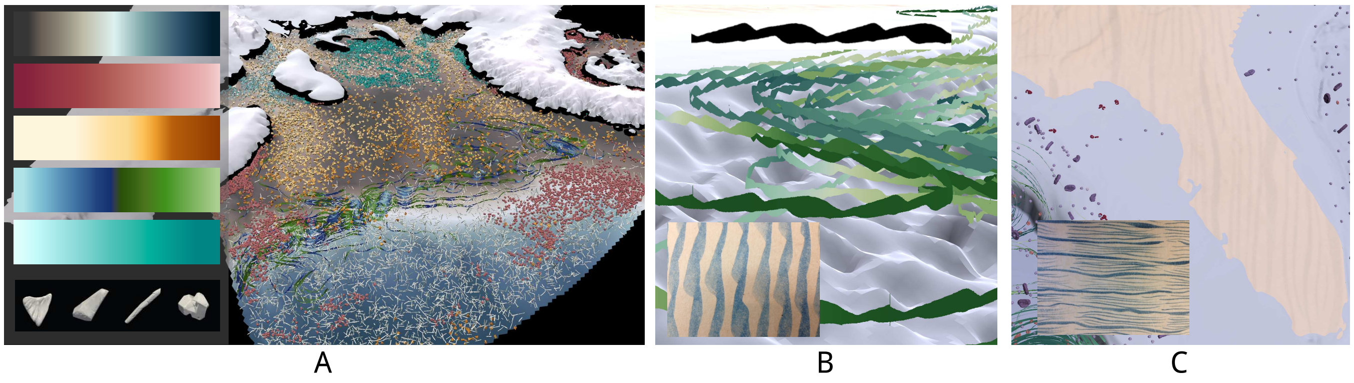

Figure 10: Francesca Samsel uses custom colormaps and her own hand-sculpted clay glyphs in this visualization of ocean currents mixing under the Ronne-Filchner ice shelf in Antarctica (A). Deborra Stewart-Pettengill commonly works with patterns, which have been digitized to form streamlines (B) in the Gulf of Mexico visualization and a texture on the land (C).

At the IEEE VIS conference in October 2020, the Sculpting Vis Collective hosted a virtual tutorial on Artifact-Based Rendering which used the architecture to facilitate collaborative remote visualization design [18]. During the tutorial, participants had the opportunity to learn about the philosophy of our team, how to use the software, and design recommendations for creating hand-crafted visualizations. Participants were then encouraged to practice with the visualization design user interface with the assistance of Sculpting Vis team members. Several example visualizations were given with the data the participants were to use, including the Antarctic visualization shown in Figure 10A. After this, participants were grouped with members of our team via video conference breakout rooms to expedite answers to questions. Nodes on a supercomputer were used to run 8 instances of the architecture for the tutorial. Each instance contained a data host, a rendering engine, and a web server for the design user interface. Tutorial participants connected to the design UI through web browsers on their personal computers and designed visualizations together with each other and the Sculpting Vis team via video conferencing software. The tutorial had approximately 40 participants total, about 10 of whom remained for the interactive portion to use the design interface.

5.2 Design User Interface Insights

From the beginning of the software design and development process, we worked with artists in our interdisciplinary Sculpting Vis Collective to determine requirements for the architecture and interface as well as enhance their functionality and form. During the formulation of the interface, the Sculpting Vis Collective held semiweekly design meetings that focused around next steps for developing the interface's capabilities as well as higher level discussions about the implications the interface and architecture will have on the visualization field. At each meeting, artists on the team discussed what was working well with the interface and what needed to change to better serve their creative processes. Even early in its evolution, the artists were excited for the possibilities that the interface allows for them while creating visualizations, including the rapid iteration through the visual possibility space and the scope and fidelity of visual vocabulary granted by the artists' use of their own media in the visualizations (Requirements 3.1.1 and 3.1.2).

The team received further valuable insights during the artist sessions which had a significant influence on the look, feel, and capabilities of the user interface and architecture. One noteworthy item during these sessions was the artists' enthusiasm for using their own work in the visualizations; this reinforced Requirement 3.1.2 as an essential item for the architecture to support. When Stewart-Pettengill first saw her 8 chine-collé textures in her palette of the design user interface, she immediately recognized them: "Oh yeah, there are my strokes! Cool!" She went on to apply several of these textures to the lines depicting ocean currents as well as the surface depicting the landmass surrounding the Gulf of Mexico (see Figure 10 B and C). Zeller also experienced a similar excitement for using her own colormaps, applying these custom colormaps to nearly every variable in her visualization. Another piece of insight was observing how the artists worked with the interface; the importance of Requirement 3.1.1, supporting an iterative design process, was affirmed by the artists. Both artists approached the visualization design using a nonlinear process – each design decision came as a reply to the current state of the visualization. Zeller found this process similar to that which she uses in her large-format paintings; she works on minute details up close and repeatedly steps back to look at the "big picture" to make sure everything is visually balanced. Stewart-Pettengill also remarked that she is "just so used to experimenting with things" as she works, and that the interface enabled a similar experimentation process during her session.

6 DISCUSSION AND FUTURE WORK

6.1 Implications for Collaborative Scientific Visualization

Our distributed architecture provides the modular separation necessary to promote the use of artistic processes while working with big data, and allows for the flexibility to host different components on geographically distributed computers. This flexibility is evidenced both by the diversity of use cases that our team has already explored using this architecture.

In total, we have held four major visualization design sessions with the architecture since its inception, all of which have taken place during the ongoing pandemic. While remote collaboration wasn't an initial requirement of the architecture's software design, it quickly became necessary to the ongoing progress of the project. Losev et al. describe a remote collaboration strategy "simulating a co-located design space" in which team members each have two webcams - one which to show the member's face and one to use showing sketches [20]. Our architecture employs a similar strategy, except instead of using a second webcam, the "sketching" and ideation on a visualization is done directly in the web-based design user interface. This is the strategy which was used during all of our design sessions thus far and it demonstrated its effectiveness for generating unique and rich visualizations like those shown in Figure 1 and Figure 10. We anticipate that even in the future beyond the pandemic, this type of remote collaboration will be incredibly useful for geographically distributed teams like the Sculpting Vis Collective.

Achieving accessibility to visualization design software at varying levels of graphics hardware is another benefit to the modular architecture. Since many actively-studied datasets are very large, it is not feasible to move them between computers; this has already been addressed with systems like ParaView [1] which allow the user to connect to another instance of ParaView running on a different computer. Our architecture provides this, in addition to two further degrees of accessibility – remote design and remote rendering of visualizations. Visualizations can be designed remotely via the artist-focused design interface, which only requires a device with a web browser and a network connection. Remote rendering allows the user to view their visualization designs without having a significant rendering capacity on their devices [14]. These two techniques are steps towards lowering the barriers for artists working with data; the hardware requirements for designing a visualization are significantly reduced when using this architecture as long as there is a data host and rendering engine available via a network.

6.2 Implications for Artistic Expression

The speed at which one can explore visual alternatives with the design interface stimulates the artistic imagination of those who use it. All of the artists who have used the interface so far have remarked that it was a fun experience for them. They also mentioned that working with the interface was very similar to their studio processes; it allows for rapid experimentation of visual alternatives as well as "taking a step back" to see the bigger picture of the visualization, which are all core principles in creative methodologies. The interface's proclivity towards visual exploration was also evidenced by the number of VisAssets used by each artist during their design sessions. The interface removes the barriers to achieving this creative state so that artists can use their skills built up over a lifetime of experience to add their voices and visions to the multidisciplinary efforts to wrangle increasingly large and complex scientific data.

The visual variation exhibited by artists using the design interface in the new architecture was significant because its workflow enables each individual to contribute their artistic vision to the visualization design. Past work with the Sculpting Vis Collective (e.g. [15, 29, 30]) and experiences with the design interface so far indicate that a crucial part of artists interacting with data is using their own handcrafted elements in their visualizations. The architecture supports this aspect of artistic engagement in visualization, as evidenced by artists' use of their own chine-collé work, custom colormaps, and hand-sculpted clay glyphs.

6.3 Future Work

The next direct step in achieving the full potential of the architecture is the integration of the JSON schema proposed in Section 4.4. The schema introduces a new level of formalization and stability the current architecture does not yet have, while still retaining the full flexibility of the artist-driven visualizations. This formalization benefits the architecture in that it ensures every component uses the same definition of a state, and it streamlines the process for creating new user interfaces on top of the architecture. This opens up the possibility for many different user interfaces to be created that all connect to the same visualization, for example a scientist-facing data manipulation interface, a simplified museum-goer interface, and a tangible user interface that integrates directly into artists' physical studio processes.

7 CONCLUSIONS

Throughout the creation of this architecture to support artists building 3D multivariate data visualizations, we have decoupled the components of current visualization software, developed user interfaces to support creative processes, connected these interfaces together with visualization rendering engines, and built a formalization of an artist-created visualization state as a JSON schema. The architecture introduces a distributed approach to visualization design software, ensuring that the specification for the rendering system is separate from the design requirements of any artist-facing user interface. Similar to previous work in artist-focused systems for visualization design, the emphasis of our work is on flexibility, both in a user interface that supports rapid iteration and in a modular architecture that allows easy modification of its software components. The Sculpting Vis Collective continues to work towards our common goal of providing artists in visualization with the best possible experience for the benefit of science and humanity at large, and the distributed architecture this paper presents is a significant step forward for artists designing 3D multivariate visualizations.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grants: 1757916, 1704604, and 1704904. MPAS-Ocean simulation data thanks to P. Wolfram, M. Maltrud, Los Alamos National Laboratory.

We would also like to thank the artists who participated in the user interface feedback sessions, and Wingate Studio for the use of their imagery.

REFERENCES

[1] James Ahrens, Berk Geveci, and Charles Law. 2005. Paraview: An end-user tool for large data visualization. *The Visualization Handbook* 717 (2005).
[2] Vinod Anupam, Chandrajit Bajaj, Daniel Schikore, and Matthew Schikore. 1994. Distributed and collaborative visualization. *Computer* 27, 7 (1994), 37–43.

[3] Bill Buxton. 1997. Artists and the art of the luthier. *ACM SIGGRAPH Computer Graphics* 31, 1 (1997), 10–11.

[4] Bill Buxton. 2010. *Sketching user experiences: getting the design right and the right design.* Morgan kaufmann.

[5] Alberto Cairo. 2013. *The Functional Art:An introduction to information graphics and visualization.* New Riders, Berkeley, CA.

[6] Hank Childs, Eric Brugger, Brad Whitlock, Jeremy Meredith, Sean Ahern, David Pugmire, Kathleen Biagas, Mark Miller, Cyrus Harrison, Gunther H. Weber, Hari Krishnan, Thomas Fogal, Allen Sanderson, Christoph Garth, E. Wes Bethel, David Camp, Oliver Rübel, Marc Durant, Jean M. Favre, and Paul Navrátil. 2012. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight.* 357–372.

[7] Donna J Cox. 1988. Using the supercomputer to visualize higher dimensions: An artist's contribution to scientific visualization. *Leonardo* 21, 3 (1988), 233–242.

[8] Donna J Cox. 1991. Collaborations in art/science: Renaissance teams. *The Journal of biocommunication* 18, 2 (1991), 15–24.

[9] Ian Fette and Alexey Melnikov. 2011. The websocket protocol.

[10] Jinzhu Gao, Jian Huang, C Ryan Johnson, and Scott Atchley. 2005. Distributed data management for large volume visualization. In *VIS 05. IEEE Visualization, 2005.* IEEE, 183–189.

[11] Ira Greenberg, Dianna Xu, and Deepak Kumar. 2013. *Processing: Creative Coding and Generative Art in Processing 2.* Apress.

[12] Tobias Isenberg, Maarten H Everts, Jens Grubert, and Sheelagh Carpendale. 2008. Interactive exploratory visualization of 2D vector fields. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 983–990.

[13] Yvonne Jansen, Pierre Dragicevic, Petra Isenberg, Jason Alexander, Abhijit Karnik, Johan Kildal, Sriram Subramanian, and Kasper Hornbæk. 2015. Opportunities and challenges for data physicalization. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems.* 3227–3236.

[14] Seth Johnson. 2020. *Palpable Visualizations: Techniques for Creatively Designing Discernible and Accessible Visualizations Grounded in the Physical World.* Ph.D. Dissertation. University of Minnesota.

[15] Seth Johnson, Francesca Samsel, Gregory Abram, Daniel Olson, Andrew J Solis, Bridger Herman, Phillip J Wolfram, Christophe Lenglet, and Daniel F Keefe. 2019. Artifact-Based Rendering: Harnessing Natural and Traditional Visual Media for More Expressive and Engaging 3D Visualizations. *IEEE transactions on visualization and computer graphics* 26, 1 (2019), 492–502.

[16] D.F. Keefe, D.B. Karelitz, E.L. Vote, and D.H. Laidlaw. 2005. Artistic collaboration in designing VR visualizations. *Computer Graphics and Applications, IEEE* 25, 2 (2005), 18–23.

[17] Daniel F Keefe, Daniel Acevedo, Jadrian Miles, Fritz Drury, Sharon M Swartz, and David H Laidlaw. 2008. Scientific sketching for collaborative VR visualization design. *IEEE Transactions on Visualization and Computer Graphics* 14, 4 (2008), 835–847.

[18] Daniel F. Keefe, Francesca Samsel, and Bridger Herman. 2020. Artifact-Based Rendering: VR Visualization by Hand. IEEE VIS Tutorial, Virtual.

[19] Scott Kildall. 2014. Data Crystals: Conglomerated World Stats. http://dataphys.org/list/data-crystals-conglomerated-world-stats/. Accessed August 2020.

[20] Tatiana Losev, Sarah Storteboom, Sheelagh Carpendale, and Søren Knudsen. 2020. Distributed Synchronous Visualization Design: Challenges and Strategies. In *2020 IEEE Workshop on Evaluation and Beyond-Methodological Approaches to Visualization (BELIV).* IEEE, 1–10.

[21] Till Nagel and Christopher Pietsch. 2015-2018. cf. city flows: A comparative visualization of urban bike mobility. https://uclab.fh-potsdam.de/cf/. Accessed August 2020.

[22] Chris North and Ben Shneiderman. 2000. Snap-together visualization: a user interface for coordinating visualizations via relational schemata. In *Proceedings of the working conference on Advanced visual interfaces.* 128–135.

[23] Erik Pasternak, Rachel Fenichel, and Andrew N Marshall. 2017. Tips for creating a block language with blockly. In *2017 IEEE Blocks and Beyond Workshop (B&B).* IEEE, 21–24.

[24] Mark R Petersen, Xylar S Asay-Davis, Anne S Berres, Qingshan Chen, Nils Feige, Matthew J Hoffman, Douglas W Jacobsen, Philip W Jones, Mathew E Maltrud, Stephen F Price, et al. 2019. An evaluation of the ocean and sea ice climate of E3SM using MPAS and interannual CORE-II forcing. *Journal of Advances in Modeling Earth Systems* 11, 5 (2019), 1438–1458.

[25] Mark R. Petersen, Douglas W. Jacobsen, Todd D. Ringler, Matthew W. Hecht, and Mathew E. Maltrud. 2015. Evaluation of the arbitrary Lagrangian-Eulerian vertical coordinate method in the MPAS-Ocean model. *Ocean Modelling* 86, 0 (2015), 93 – 113. https://doi.org/10.1016/j.ocemod.2014.12.004

[26] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. 2016. Foundations of JSON schema. In *Proceedings of the 25th International Conference on World Wide Web.* 263–273.

[27] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

[28] Francseca Samsel, Gregory Abram, Seth Johnson, Bridger Herman, and Daniel F. Keefe. [n.d.]. The Sculpting Vis Library. http://sculpting-vis.tacc.utexas.edu/library.

[29] Francesca Samsel, Annie Bares, Seth Johnson, and Daniel F. Keefe. 2019. Scientific Visualization: Enriching Vocabulary via the Human Hand. IEEE VIS Arts Program.

[30] Francesca Samsel, Lyn Bartram, and Annie Bares. 2018. Art, Affect and Color: Creating engaging expressive scientific visualization. In *2018 IEEE VIS Arts Program (VISAP).* IEEE, 1–9.

[31] Francesca Samsel, Sebastian Klaassen, and David H Rogers. 2018. Colormoves: Real-time interactive colormap construction for scientific visualization. *IEEE computer graphics and applications* 38, 1 (2018), 20–29.

[32] Francesca Samsel, Phillip Wolfram, Annie Bares, Terece L Turton, and Roxana Bujack. 2019. Colormapping resources and strategies for organized intuitive environmental visualization. *Environmental Earth Sciences* 78, 9 (2019), 269.

[33] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 341–350.

[34] Denise Schmandt-Besserat. 1999. Tokens: the cognitive significance. *Documenta Praehistorica* 26 (1999), 21–27.

[35] David Schroeder, Dane M Coffey, and Daniel F Keefe. 2010. Drawing with the flow: a sketch-based interface for illustrative visualization of 2d vector fields.. In *SBM.* Citeseer, 49–56.

[36] David Schroeder and Daniel F Keefe. 2015. Visualization-by-sketching: An artist's interface for creating multivariate time-varying data visualizations. *IEEE transactions on visualization and computer graphics* 22, 1 (2015), 877–885.

[37] Will J Schroeder, Bill Lorensen, and Ken Martin. 2004. *The visualization toolkit: an object-oriented approach to 3D graphics.* Kitware.

[38] Robert W Schwanke. 1991. An intelligent tool for re-engineering software modularity. In *Proceedings-13th International Conference on Software Engineering.* IEEE Computer Society, 83–84.

[39] Adrien Segal. 2015. Grewingk Glacier. https://www.adriensegal.com/grewingk-glacier. Accessed July 2020.

[40] Ben Shneiderman. 2000. Creating creativity: user interfaces for supporting innovation. *ACM Transactions on Computer-Human Interaction (TOCHI)* 7, 1 (2000), 114–138.

[41] Ben Shneiderman. 2002. Creativity support tools. *Commun. ACM* 45, 10 (2002), 116–120.

[42] Ben Shneiderman. 2004. Designing for fun: how can we design user interfaces to be more fun? *interactions* 11, 5 (2004), 48–50.

[43] Ben Shneiderman. 2007. Creativity support tools: Accelerating discovery and innovation. *Commun. ACM* 50, 12 (2007), 20–32.

[44] Gary Singh. 2018. Wearing Multiple Hats. *IEEE Computer Graphics and Applications* 38, 4 (2018), 6–8.

[45] Marc Swackhamer, Andrea J Johnson, Daniel Keefe, Seth Johnson, Ross Altheimer, and Aaron Wittkamper. 2017. Weather report: Structuring data experience in the built environment. *Proceedings of Architectural Research Centers Consortium* (2017), 102–111.

[46] Michael Terry and Elizabeth D Mynatt. 2002. Side views: persistent, on-demand previews for open-ended tasks. In *Proceedings of the 15th annual ACM symposium on User interface software and technology.* 71–80.

[47] Nicola Valcasara. 2015. *Unreal engine game development blueprints.* Packt Publishing Ltd.

[48] Maarten Van Steen and Andrew S Tanenbaum. 2017. *Distributed Systems* (3rd ed.). Maarten van Steen, Pearson Education Inc.

[49] Mythravarun Vepakomma. 2014. *Blender Compositing and Post Processing.* Packt Publishing Ltd.

[50] Fernanda B Viégas and Martin Wattenberg. 2007. Artistic data visualization: Beyond visual analytics. In *International Conference on Online Communities and Social Computing.* Springer, 182–191.

[51] Yun Wang, Adrien Segal, Roberta Klatzky, Daniel F Keefe, Petra Isenberg, Jörn Hurtienne, Eva Hornecker, Tim Dwyer, and Stephen Barrass. 2019. An emotional response to the value of visualization. *IEEE computer graphics and applications* 39, 5 (2019), 8–17.

[52] C. Ware. 2012. *Information Visualization: Perception for Design.* Morgan Kaufmann Publishers, San Francisco, CA.

[53] Phillip J Wolfram, Todd D Ringler, Mathew E Maltrud, Douglas W Jacobsen, and Mark R Petersen. 2015. Diagnosing isopycnal diffusivity in an eddying, idealized midlatitude ocean basin via Lagrangian, in Situ, Global, High-Performance Particle Tracking (LIGHT). *Journal of Physical Oceanography* 45, 8 (2015), 2114–2133.

[54] Nick Qi Zhu. 2013. *Data visualization with D3. js cookbook.* Packt Publishing Ltd.