ORIGINAL ARTICLE

CERVI: collaborative editing of raster and vector images

Ulrike Bath[1] · Sumit Shekhar[1] · Julian Egbert[1] · Julian Schmidt[1] · Amir Semmo[2] · Jürgen Döllner[1] ·
Matthias Trapp[1]

Abstract

Various web-based image-editing tools and web-based collaborative tools exist in isolation. Research focusing to bridge the gap between these two domains is sparse. We respond to the above and develop prototype groupware for real-time collaborative editing of raster and vector images in a web browser. To better understand the requirements, we conduct a preliminary user study and establish communication and synchronization as key elements. The existing groupware for text documents or presentations handles the above through well-established techniques. However, those cannot be extended as it is for raster or vector graphics manipulation. To this end, we develop a document model that is maintained by a server and is delivered and synchronized to multiple clients. Our prototypical implementation is based on a scalable client–server architecture: using WebGL for interactive browser-based rendering and WebSocket connections to maintain synchronization. We evaluate our work qualitatively through a post-deployment user study for three different scenarios. For quantitative evaluation, we perform a thorough performance measure on both client and server side, thereby identifying design recommendations for future concurrent image-editing software(s).

1 Introduction

Collaboration between visual artists dates back to as early as late sixteenth century (Fig. 1a). In olden times, this process was completely manual where two or more artists specializing in different genres would physically meet to create a

✉ Sumit Shekhar
sumit.shekhar@hpi.uni-potsdam.de

Ulrike Bath
ulrike.bath@student.hpi.uni-potsdam.de

Julian Egbert
julian.egbert@student.hpi.uni-potsdam.de
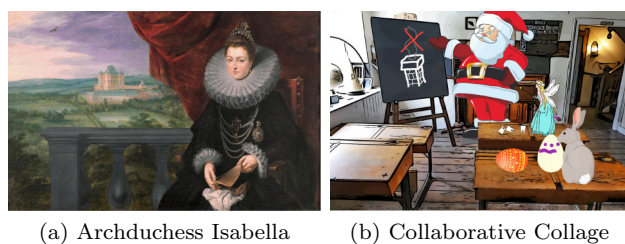
Julian Schmidt
julian.schmidt@student.hpi.uni-potsdm.de

Amir Semmo
amir.semmo@digitalmasterpieces.com

Jürgen Döllner
jurgen.dollner@hpi.uni-potsdam.de

Matthias Trapp
matthias.trapp@hpi.uni-potsdam.de

[1]   Hasso Plattner Institute, University of Potsdam, Potsdam,
      Germany

[2]   Digital Masterpieces GmbH, Potsdam, Germany

shared painting. In the modern era, the above practice continued resulting in various masterpieces [15]. However, its adaptation in the digital domain is progressing only slowly (Fig. 1b). Even though there exist collaborative applications mimicking a shared whiteboard—allowing for doodling and/or simple manipulations of a shared image. A system for real-time collaborative editing of raster or vector images at different levels of functionality or control—similar to common image editing desktop applications (e.g., Adobe Photoshop or GIMP)—does not exist to the best of our knowledge [13].

This work focuses on the concept and implementation of a web-based real-time collaborative editing application for raster and vector images. It supports a range of image manipulations that enables multiple users to collaboratively edit both image representations interactively. The intuitive user interface (UI) aids in mitigating the risk of access conflict as participants modify the same data. The application provides a responsive graphical user interface (GUI) that facilitates access using mobile devices, such as smart phones or tablets.

For it, we choose the following approach. Section 2 states related requirements and challenges on graphics collaborative editing and describes use-cases obtained by a user

(a) Archduchess Isabella   (b) Collaborative Collage

Fig. 1 **a** An example of collaboration between *Jan Brueghel the Elder* and *Peter Paul Rubens* approx. 1616-17. **b** A collage created collaboratively using our web-based system. It comprises the blending of multiple raster and vector layers, vector strokes, and image processing operations (e.g., cartoon effect in the background)

study regarding the current state of real-time collaborative image editing and associated tasks. Section 3 reviews and analyzes related work and existing tools on collaborative editing of graphics. Section 4 describes a system overview of our implementation based on server and client functionality. Section 5 evaluates the implementation through a post-deployment user study and performance analysis. We summarize our findings and potentials for future work and research in Sect. 6.

2 Problem statement

This section reviews major requirements for a collaborative system (Sect. 2.1), identifies fundamental challenges for collaborative image editing (sect. 2.2) and reports on a preceding user study (Sect. 2.3), conducted to analyze potential conflicts that needs to be addressed (sect. 2.4).

2.1 Basic system requirements

Multi-user systems where the actions of one user must quickly be propagated to the other collaborator are referred to as real-time groupware [7]. During recent years, various instances of such systems emerged into today's distributed, collaborative working environments. For example, systems such as Google Workspace or Microsoft Office 365 Online edition, various massively multiplayer online games, or NVIDIA's Omniverse platform for 3D contents. All such systems have in common that (1) a document instance or a shared context is hosted by a server(s), is (2) synchronized using a service, and (3) can be manipulated by multiple participants. The most relevant characterizing aspects of a real-time groupware system, according to Ellis and Gibs [8], are as follows:

> Interactive and real-time (Aspect-1): i.e., response times must be short and notification times must be comparable to response times.

> Distributed (Aspect-2): i.e., in general, one cannot assume that the participants are all connected to the same machine or even to the same local area network.
> Volatile and Ad-hoc (Aspect-3): i.e., participants are free to come and go during a session and generally are not following a pre-planned script. It is not possible to tell a priori what information will be accessed.
> Focused (Aspect-4): i.e., during a session there is high degree of access conflict as participants work on/modify the same data.

A popular application that fulfills all of the above criteria is Google Docs [10]. The cloud-based service provided by Google has revolutionized the way people edit documents collaboratively. However, when editing text, most standard algorithms do not consider the complete structure of the document and make use of per-line diffing and merging.

2.2 Fundamental challenges

The above approach for documents cannot be extended for images in a straightforward manner: while Aspect-1 and Aspect-2 largely pose specific technical challenges (e.g., undo/redo functionality and latencies), Aspect-3 and Aspect-4 reflect on the spatial, structural, and temporal features of collaborative raster-image editing. Existing collaborative whiteboard applications maintain their state by tracking the brush strokes of individual clients [1,5]. They allow users to doodle/sketch on top of the image but hardly provide any tools for image editing itself. On the other hand, existing web-based image editing tools are not collaborative [18,19]. A system that is quite close to what we aspire is the Google Draw, a functionality provided as part of Google Workspace [11]. Even though it allows users to collaboratively edit attributes of a shared image, the range of per-pixel edits is limited.

We adopt a human-centered design process to identify the challenges associated with a real-time collaborative image editing system. To this end, we design and conduct a preceding user study based on a questionnaire. The answers to the above questionnaire identify key design principles.

2.3 Preceding user study

For better understanding of the design rationals for a real-time collaborative image-editing application, we conducted a preceding user study. Subsequently, we analyzed results and major findings, identified main use cases, and created GUI design sketches.

Participants and Study Design. We selected participants who have experience in collaborative image editing with existing technologies. They belong to a broad range of background and have performed image editing: for casual creativity

and/or as a professional activity. A total of 27 participants answered the 17 questions. The questions broadly addressed the following aspects: (*i*) *How do you perform collaborative image editing tasks?* and (*ii*) *What are the challenges associated with it?*. The challenges thus identified are used as the basis for designing our system.

Summary on Challenges in Collaboration. The foundation of any collaborative task is efficient *communication*, which also reflects in our survey answers: "*Communication is everything, it is sometimes hard to get an artistic idea thru*". "*Communicating who edits what and how*". The lack of communication is not only restricted to high-level requirements and task sharing but also low-level details such as data/edits synchronization: "*Staying in sync, keeping a history of changes, knowing what the partners already have done*". "*Handling data conflicts, know on which parts or region my teammates are currently working on, handling different versions...*". To mitigate the above problems, users make use of existing communication channels. However, such an approach seems to be quite inefficient in terms of both data and time: "*It takes a lot of time sending images back/forward and see when progress is made*". "*Sharing huge files of raw pictures with the team and keeping them in sync*".

Design Inference. Concerning the above challenges, we provide an integrated messaging functionality within our system enabling efficient communication. Our WebGL-based rendering framework allows for image edits that are visible in real-time among the collaborators, further enabling low-level communication. Our system supports a range of application scenarios where collaborative image editing can be used: "*logo creation*", "*poster designing*", "*creative editing*", etc.. We support such varying application scenarios by allowing editing of raster and vector graphics, mouse/touch-based sketching interface, along with visual computing asset (VCA)-based raster-image manipulation. The user edits are maintained as part of session management providing a consistent editing environment. Data conflicts are handled with complementary update processes on server and client-side (Sect. 4.1).

2.4 Potential conflicts in collaborative editing

There are several potential conflicts arising in real-time collaborative image editing systems, especially if these support a variety of tools being applied to multiple layers. Specific to our approach this concerns challenges arising from (*i*) limited attention-bandwidth and (*ii*) synchronous as well as (*iii*) asynchronous editing conflicts. Considering users operating in the same sessions, our system offers tools to approach the above challenges.

Limited Attention-bandwidth. While performing editing tasks, such as painting or designing, a user focuses on the immediate effect of the current tool. Thus, the user has a limited attention-bandwidth and is usually not aware of the changes performed by other users in the document. For example, adding new layers impacts the layer order and often interrupts the user's workflow, and can quickly lead to confusion. To counterbalance this, a document version history is offered that enables users to comprehend the performed changes over time.

Synchronous Editing Conflicts. There are various causes for conflicts in synchronous data editing, e.g., users use the same/different tool on a given layer or a layer is about to be removed that is currently used by other users. Instead of making tools modal, we choose to raise awareness by indicating that another user is using the same tool or has selected the same data using visual feedback. For this purpose, colored hints (lines) visualize which user(s) currently select a layer and tool, respectively (Sect. 4.3.3). Moreover, the cursors of all users are depicted in the respective avatar color (Sect. 4.1.2).

Asynchronous Editing Conflicts. These conflicts are often caused if several users simultaneously edit the same layer or due to interruptions of unfinished tasks, e.g., a user is interrupted in its current workflow but wants to continue his/her work later on. To resolve the above, we introduce an *exclusive-lock* functionality for a layer, i.e., a user can forbid editing of a layer for everyone except himself/herself (Sect. 4.3.4). To avoid deadlock scenarios, a layer can be *exclusive-unlocked* by others users. In this case, the user who initiates the exclusive-lock is notified accordingly.

3 Background & related work

The challenges associated with collaborative image editing have two aspects: the conceptual, design, and implementation level, which nowadays demands web-based approaches using services. The existing web-based applications mainly address sketching and/or designing functionalities.

3.1 Collaborative graphics editing

An analysis with respect to the pertinent properties of a collaborative graphics editing system was first done by Sun and Chen [23], where the authors present a formal blueprint for versioning, consistency maintenance, and conflict resolution. Heer and Agrawala [12] investigate visual analytic tools and present approaches for increased user engagement and better shared context and awareness. As a particular example for such systems, Salvati *et al.* [22] and Calabrese *et al.* [3] propose ideas regarding real-time collaborative mesh manipulation via version history merging and robust

sharing. Gao *et al.* [9] remodel the 2D graphical operations as linear operations by mapping the two-dimensional drawing area into a linear structure, for the purpose of collaborative image manipulation. For preventing conflicts related to consistency, Wu *et al.* [26] present a generic Common Graphics Collaborative Editing (CGCE) algorithm. To showcase their methods, both Gao *et al.* and Wu *et al.* choose current web-based technologies, however, as a functionality they only focus on sketching or primitive geometric shape editing. Zhai *et al.* [27] construct an approach for collaborative image modification on mobile phones using wireless communication. However, only straightforward granular operations—of *import, export, update*, and *commit*—is dealt with their method. An application meant explicitly for the purpose of architectural communication via cooperative sketching is developed by Novakova *et al.* [17]. In a recent work, Bath *et al.* [2] present a system for collaborative editing of raster images. In contrast to most of the above approaches, our system provides a broad range of image edits, includes rendering of both raster and vector layer(s), and also has provision for coloring and/or sketching.
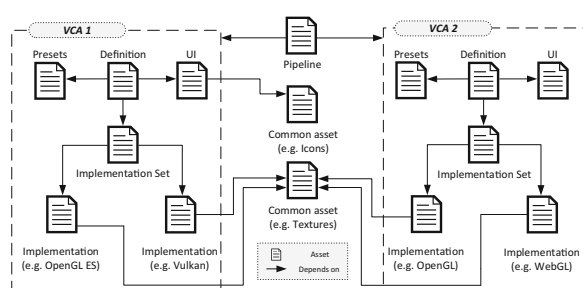
3.2 Image processing as a service

For implementing image processing techniques (e.g., filtering or stylization) for web-based applications, one can distinguish between client-side or server-side image processing approaches [14]. Server-side implementation based on microservices have the advantage of increased scalability, easy deployment and maintainability as well as, the possibility to introduce various technologies into one system [24]. However, high update latencies limit their usage in real-time applications [21,25]. Therefore, we favor the integration of client-side processing for our approach by developing a WebGL-based image processor. With respect to modeling image processing techniques, Dürschmid *et al.* [6] present an approach that supports collaborative design of stylization

techniques for mobile devices. We adopt their concept of representing and sharing image and video processing operations, further denoted as VCAs for our prototype.

A VCA represents data and control logic of image processing operations and provides an interface that abstracts from specific implementations. This platform-independent presentation facilitates the reuse of building blocks based on document modularity. An structural overview of the respective concept, comprising two VCAs with shared resources, is depicted in fig.2. It consist of the following document types. The *Definition* represents an "interface" describing the parameters (type, range, default values) of an image processing operation. It references (1) multiple *Preset* files that define grouped parameter values, (2) *user interface* definition of resources for GUI purposes, and (3) an *Implementation Set*, i.e., the definition of an VCA implementation mapping to target-specific implementation. Further, a *Pipeline* enables the successive combination of multiple VCAs.

3.3 Web-based sketching and designing

Nowadays, visual ideation is made possible in a collaborative setting via web-based shared whiteboards e.g., Aggie.io or Draw.chat [1,5]. More recently, online design-tools also allow for collaborative creation of designs e.g., Figma, Canva, or AdobeXD. Nonetheless, both kinds of existing web-based tools hardly provide any support for image processing. Pixel-level global and local processing of images is enabled by another set of online applications which are not collaborative in nature e.g., Photopea, Pixlr, (Adobe-)Photoshop Web [18,19]. Limited collaboration, in an asynchronous manner, for raster and vector images is made possible by recent tools from Adobe Creative Cloud. In comparison, we aim to create a web-based system that provides real-time synchronous collaboration. Table 1 compares existing web-based photo-editing and whiteboard applications regarding the following aspects:

Layer (Yes/No): The application does support layering of multiple images. This allows for an increased function scope and assumes a complex data model.
Direct Manipulation (Yes/No): The application does support direct manipulation of image contents, e.g., using coloring or transform functionality.
Undo/Redo (Yes/No): The support of undo/redo functionality facilitates error-tolerance while using direct manipulation metaphors.
Image Filtering (None/Destructive/Non-destructive): An application supports the usage of single or multiple destructive/non-destructive image filtering operations.
Data Type (Raster/Vector/Both): The application can handle raster, vector, or both types of input data.

Fig. 2 Structural overview of VCAs by example. VCA 1 is implemented for the two platforms OpenGL ES and Vulkan. The specific render pipelines are described in the respective effect implementation files. Due to the asset separation, all implementations of VCA 1 and VCA 2 can reference and reuse the same set of textures

Table 1 Comparison of various web applications for image editing, with respect to different features

Application	Layer	Direct Manip.	Undo/Redo	Image Filtering	Data Type	Resp. GUI	Collaboration
canvaspaint.org	No	Yes	Yes	None	Raster	Yes	None
pixlr.com	Yes	Yes	Yes	Destructive	Raster	No	None
photopea.com	Yes	Yes	Yes	Destructive	Raster	No	None
draw.chat	No	Yes	Yes	None	Vector	No	Synchronous
aggie.io	Yes	Yes	Yes	None	Raster	Yes	Synchronous
Google Draw	No	No	No	Non-destructive	Raster	Yes	Synchronous
Adobe Creative Suite	Yes	Yes	Yes	Destructive	Both	Yes	Asynchronous
COLiER	Yes	Yes	Yes	Destructive	Raster	Yes	Synchronous
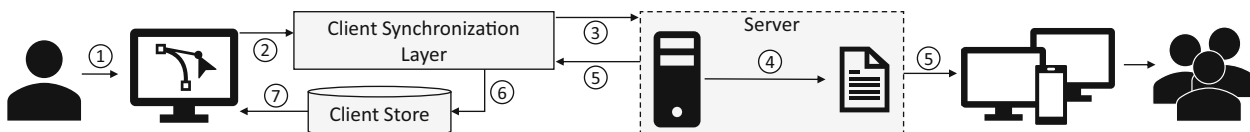
Fig. 3 Sequence of client–server communication: (1) user modifies project, (2) modified parameters are processed in client's synchronization layer, (3) a change request is sent to server, (4) server updates document (5) if successful, updates are sent to all clients, (6) synchronization layer updates the client store, (7) changes are applied

Responsiveness (Yes/No): The GUI of the application support responsive layout of components, thus supports desktop and mobile devices with varying screen sizes. Collaboration (None/Synchronous/Asynchronous): A collaborative application enables multiple clients to modify image data simultaneously. This requires communication between clients and modeling of messages reflecting the editing process.

Our web-based collaborative system provides sketching and designing functionality along with image manipulations. Moreover, we enable synchronous collaboration among users. It provides all features compared in Table 1 while operating on layered raster or vector images.

4 The CERVI system

We developed our system as a single-page application (SPA) that can be used on desktop systems and on mobile devices. It enables sketching, image adjustments, and creation of image
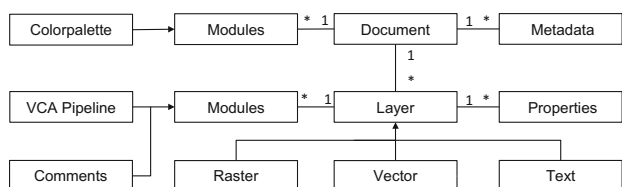
Fig. 4 High-level structure of a session document comprising multiple layers with multiple VCAs

collages among multiple clients in real time. An overview of our system with a depiction of a client to server communication and vice versa is presented in Fig. 3.

To achieve this distributed structure of a real-time groupware system, we develop an extensible client–server architecture (Fig. 5). The server is mainly responsible for session handling and synchronization, and maintaining communication among clients (Sect. 4.1). The client transmits and consumes messages (Sect. 4.2), which represent editing actions and perform client-side rendering (Sect. 4.3)

4.1 Server components and functionality

The main task of the server component is to maintain the session documents, manage and provision its state, and handle the communication between the clients.

4.1.1 Session document

The document structure (Fig. 4) is inspired by the OpenRaster file format [20]. A document consists of metadata about the project, e.g., creation date, version, resolution, etc.. All images are organized as layers and stored as arrays. We currently support raster and vector graphics as layers. However, our modular approach allows easy integration of additional layer types e.g., a text layer. These layers comprise generic information, such as geometric transformations, visibility, etc., as well as layer specific details. Both on document and layer level, the functionality can be extended through modules e.g., global color palette on document level and VCA module on layer level. The VCA references and its parame-
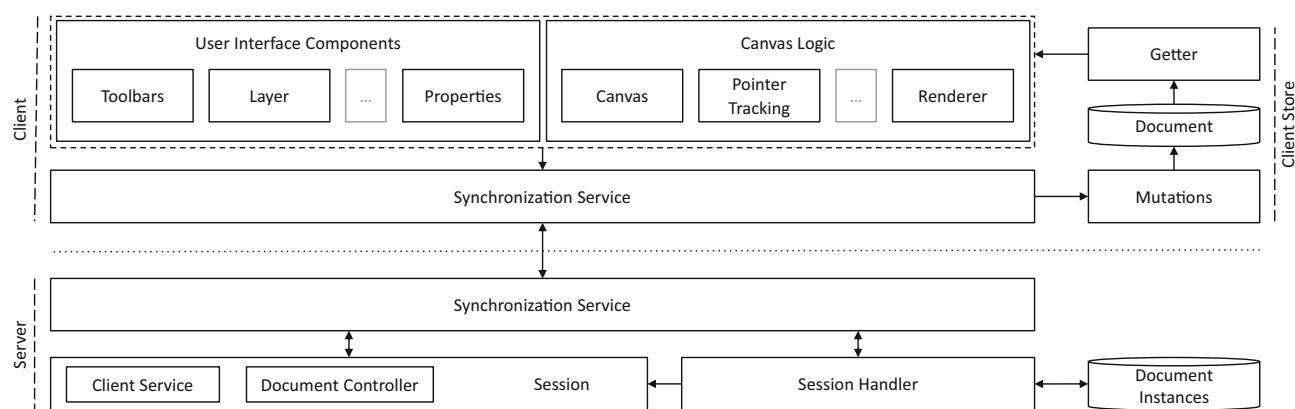
Fig. 5 Overview of the client–server architecture used for our system. The server synchronizes and propagates project modifications among clients through different session handlers. A user can modify the project through theGUI components and the content view. The cyclic update process on the client-side prevents version conflicts

ters are stored on a per-layer basis and enables further editing of raster images.

4.1.2 Session handling

At the initial state, each stored document is read into a session and is assigned a unique identifier. All sessions are subscribed to the socket service which handles the client communication. When a client connects to the server, the server responds with a session overview. After registering for a session, the user's socket is subscribed to all events of the specific session on the server-side. By registering to the server, the client receives its unique server-socket Identifier (ID) that is stored in the local storage of the browser. If the client disconnects, it re-sends its assigned ID when reconnecting to the server and thus is recognized again. Moreover, a unique color is assigned to the client, which also serves as the default brush color.

Since several users can work simultaneously in one session, we have a high degree of access-conflict. The server treats incoming changes as "first come, first serve" and, hence, defines the order of updates which is then sent to all subscribed clients. However, if several changes are requested simultaneously the latest change is eventually displayed as it would overwrite the earlier requests. The main logical conflicts are resolved at server-side, e.g., if a user deletes a given layer while another user edits it, the latter change request is dropped. Remaining access-conflicts, which are not mutually exclusive, are then handled at the client-side, i.e., the last executed update will define the modified session state. Thus, session-handling is important to maintain synchronization among clients, a key requirement (Sect. 2.3) for such a system.

4.2 Protocol for client–server communication

For communication between the server and multiple clients, we design a simple protocol that suffices the following requirements: (*i*) it has a simple yet extensible message structure to facilitate efficient development and easy integration of future features; (*ii*) it is suitable for fast message (de)serialization to reduce the run-time overhead for clients and server. The clients employ a WebSocket connection to send events to the server, which are then broadcasted to the remaining clients. Both client and server listen for events and process the incoming data accordingly. The sent data include information about the applied project changes as well as other aspects such as timestamp and client ID which allows for change-history maintenance. An exemplary message structure (based on JavaScript Object Notation(JSON) standard format) event, for a new drawn path which sends a `layer_specifics`, is depicted in List. 1. The above allows for efficient communication among clients, which is a necessity (Sect. 2.3) for our system.

```
{"tool": "brush",
"action": {
"type": "new",
"path": {
  "timeStamp": "1617804631471",
  "clientId":  "m82pY9bvAeIAAAH",
  "color":     "#795EB3",
  "width":     "10",
  "path": [["M",446.99,38],
          ["Q",447,38,448,38],
          ["Q",449,38,449.5,38],
          ["Q",450,38,451,38.5],
          ["Q",452,39,452.5,39],
          ["L",453.01,39]]}}}
```

Listing 1 Exemplary message structure of a `layer_specifics` action with the `brush` tool.
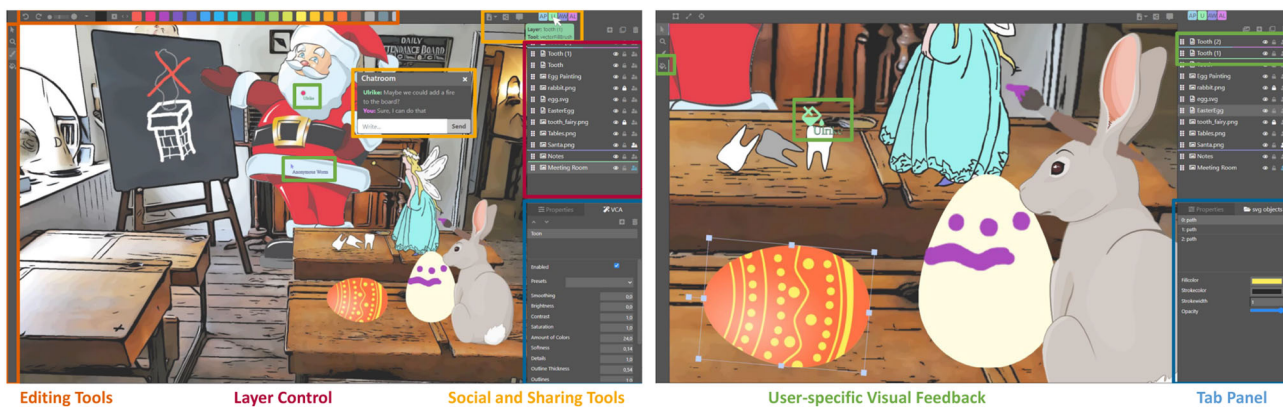
Fig. 6 Our GUI provides a variety of editing tools and user specific visual feedback to visually communicate the tool and objects currently operated by other users in order to mitigate the risk for potential editing conflicts

4.3 Client components and functionality

The rendering of the raster and vector images is performed entirely on the client-side using WebGL 2.0. The front-end is developed using Vue, and the Vuex framework is employed for global storage. To facilitate layer control for canvas rendering, we make use of Fabric.js.

4.3.1 Update logic

Since all users can potentially work on the session document simultaneously, resulting data-conflicts are required to be handled properly. For it, we propose the communication process among clients as depicted in Fig.3. After the user changes a parameter, a change-request is sent to the server over a synchronization service. The above service also listens to all change-events from the server. If the request is accepted by the server, it will notify all clients. The service then modifies the store, and the GUI is updated accordingly. The user is not allowed to update the local store directly to prevent version conflicts. A small update delay is barely noticeable because of quick socket communication. In case of a parameter-update conflict, when two users update the same value simultaneously, the last request processed by the server is considered the final version. However, only one user at a time should be able to directly manipulate or transform a layer. For it, during such operations, the layer will be implicitly locked with respect to its transformation properties.

4.3.2 GUI structure/schematics

We assume that the target audience is familiar with raster-image editing software and therefore decide to re-use GUI concepts from common image-editing applications. Thus, tools such as brush and selection are located in a vertical icon toolbar on the left with additional control parameters on an upper horizontal bar (Fig. 6). The object property panels (e.g., of layers or their properties) are both located on the right side of the canvas. A user can directly interact with the canvas by drawing on a layer or transforming it. Since the image takes up most of the available space for direct editing, the remaining GUI components are arranged compactly with informative icons to ensure intuitive usability. For smaller screen sizes, e.g., mobile or tablet, this is problematic because many operations must be clearly represented with large buttons for easy access. Therefore, we hide certain components using a responsive layout, which are displayed if required (Fig. 7). Since the components themselves do not differ between screen sizes, the user can easily switch between desktop and mobile devices without adapting to a new GUI. The generic project tool buttons for downloading the final image or specific layers 📥, sharing the project ⎘, or messaging other users 💬, are placed on top of the editing components.

4.3.3 User-specific visual feedback

For a coordinated workflow among the clients, the respective selected layer, VCA and the tool of each user is highlighted (Fig. 7(b)). This allows a user to reproduce canvas changes made by another user. Moreover, this potentially avoids editing conflicts or parameter overwrites as the user can see if someone else has selected the same layer or VCA. Similar to other collaborative web-apps, an overview of currently active users is depicted in the upper right corner. On hovering over the user's icon, the respective username is displayed. We can also get an overview of the user's working area by clicking on the user icon. The cursor position on the canvas is broadcasted to the remaining clients and is displayed with the client's unique color identifier, assigned by the server. The displayed cursor depends on the user-selected tool and can be, among other things, a pointer (for selec-
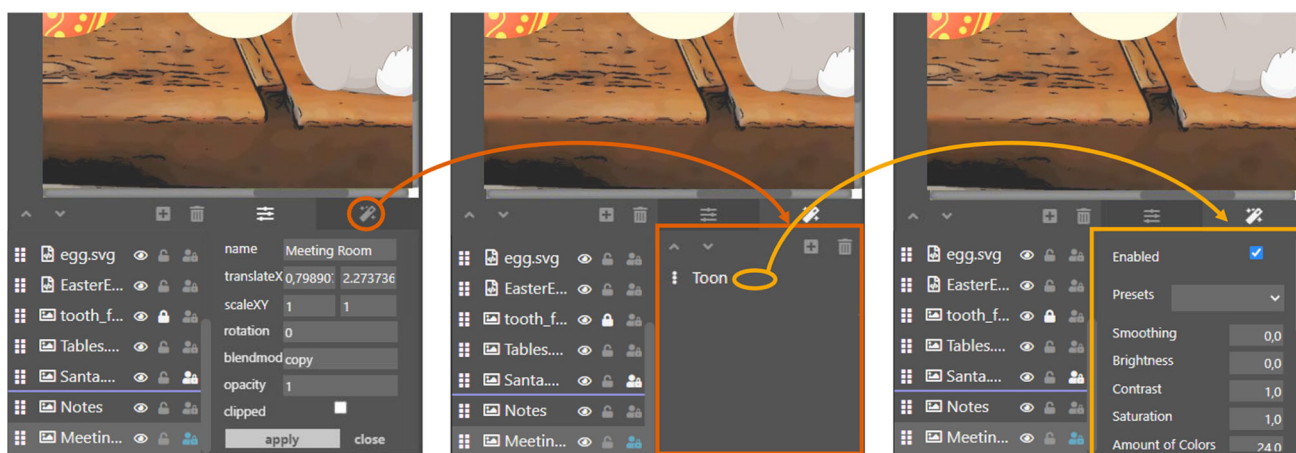
Fig. 7 The responsive GUI layout hides editing components if the screen size is too small. The functionality can be easily expanded by pressing the respective button

tion tool) or a brush (for coloring tool). Additionally, if a user selects or transforms a layer, it is highlighted with the respective client's color. This way, all participants obtain an overview of currently active objects.

4.3.4 Basic editing features

Layer Functions. Within a project, layers can be added, deleted, and reordered with simple button clicks in the layer control panel. An icon regarding the respective layer type (raster/vector) is shown next to each layer name for a better overview. Moreover, if required we also allow a vector layer to be rasterized. For each layer, visibility can be set 👁 and the layer itself can be locked/unlocked 🔒. To further enable collaboration we introduce an *exclusive-lock* button 👥🔒. Analogous to the lock functionality, a user can disable a layer via this button. However, the layer will be locked for everyone except this user. Other users can see who exclusively locked a layer and when. By unlocking this layer, the original user gets a notification. This way, a user can personally lock a layer and signal that he/she does not want interference from other users. Depending on whether the user himself/herself exclusively locked the layer or not, this button is highlighted in a different color. Thus, the user also has a visual overview of which layers he/she is currently working on.

Furthermore, for each layer, additional information is displayed in the panel below. A user can switch between the different control settings through tabs, e.g., the layer properties or VCA. The main layer properties, e.g., scale, rotation, opacity, are located in the properties tab. Since different layer types offer different functionality, the tab choices may differ accordingly. For example, the VCA tab is an add-on only for raster layers, thus, it is only displayed then. Moreover, this panel can easily be extended with additional editing features by adding new tabs.

Raster Image Functions. Raster image layers are provided by adding source images or single colored layers, both of which can be modified using the brush tool. Here, brush size and color are adjustable using respective components in the horizontal bar. Performed brush strokes can be undone/redone per layer using the respective tool buttons. This logic works for each client separately, e.g., an agent can undo or redo the own strokes after a different agent drew as well. Since each action has a timestamp, redone strokes will be applied on the layer in the original order they were first drawn. Thus, each client can undo his strokes indefinitely, the stroke with the latest timestamp on the stack will be removed and cached on a redo stack. The redo stack is cleared as soon as the client starts drawing again.

As described previously, VCAs can be applied on raster layers. In the VCA tab on the right side, the user can add, delete, and reorder VCAs in the pipeline of the layer. Each VCA is adjustable and can be enabled or disabled. All changes are applied to the image in real time.

Vector Image Functions. Similar to scalable vector graphics (SVG), a vector layer can contain different object types, e.g., paths or circles. All objects are accessible in a list in the corresponding tab on the right panel. Upon object selection, further properties such as fill or stroke colors can be modified. Vector objects can also be filled using direct manipulation on the canvas by selecting the fill tool in the toolbar. In case the user wants to change only the fill style of an object similar to a coloring book application, the colorbook mode can be enabled in the horizontal bar—preventing coloring of strokes.

4.3.5 Feature extensions

Since our application should serve different purposes, the focus on necessary features may shift. This is why we emphasized on modularity in the GUI and system architecture. Our application can easily be extended on different levels: (1)

new layer types, (2) layer plugins, and (3) document plugins. As a new layer type, the approach for vector layers can also be extended for XML3D elements. Additional raster image editing features can be added in form of a new VCA. On the document level, we showcase an efficient integration of a color palette plugin which is shared and updated with all clients.

5 Evaluation

We develop our collaborative framework in a two-step process. Firstly, we create the necessary functionality for sketching/coloring and editing of raster images, after which we perform a usability study (sect. 5.1) to understand the interesting elements of the system from a user's perspective. Secondly, based on the popularity of *coloring* and *collage creation* tasks, we further extend the system to also allow for editing of vector images (sect. 5.2). A thorough performance analysis of our complete system (Sect. 5.3) identifies challenges in terms of latency and further scalability.

5.1 Post-deployment user study

Additional requirements on functionality and user experience are often identified after a prototype is deployed and users have had a chance to try the software and provide feedback. This valuable feedback motivated us to extend our system for vector images (sect. 5.2) and will potentially be useful to improve the future iterations of our prototype. For the post-deployment study, we focused on the following three aspects: (i) do users understand the visualization metaphors to avoid editing conflicts, (ii) do users understand the general structure of the GUI, and (iii) are users satisfied with the prototype.

5.1.1 Participants & apparatus

We recruited 16 volunteers (8 male, 8 female) in 6 different groups. The above participants use our system for the first time and were not part of the preliminary user study to avoid any inherent bias. Each group had a variable number of participants between 23 and volunteers were aged between the ages 21 and 34. While all of them had experience with computers, 5 had no or only little experience with image editing applications. All of them had normal or corrected-to-normal vision and no known visual impairments. All the participants (except for one, who used an iPad) accessed our SPA on a desktop/laptop system with a single monitor using standard web-browsers (Google Chrome: 7, Mozilla Firefox: 5, Apple Safari: 2, Microsoft Edge: 2) and a computer mouse (two participants used trackpads).

We conducted a supervised/observed study in remote sessions, each with a group of participants. We were connected with them via an online Zoom meeting as they were guided and monitored at the same time. Each session had a length of approx. 60 min having the following structure. First, each group received a brief introduction into the GUI covering only editing tools as well as layer and VCA controls (5 min). Following this, each group is asked to collaboratively solve three tasks in sequence.
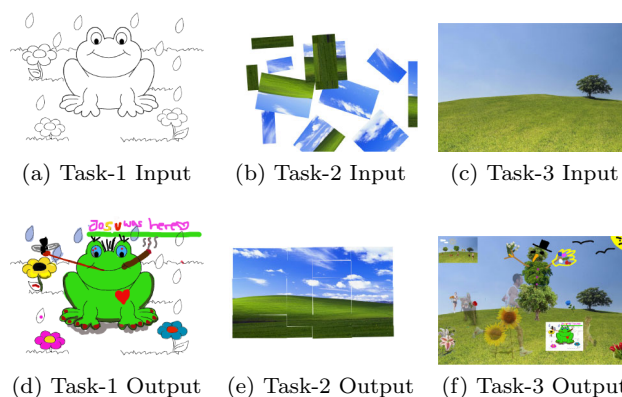
5.1.2 User tasks

The three tasks performed by each participant group cover the full potential of our editing system. The tasks are ordered by increasing difficulty and took 15min to 20min, respectively, for completion. Figure 8 shows selected results obtained during the study.

Coloring (Task-1). We provide a blank sketch as a background layer (Fig. 8a) and the participants are asked to color the sketch using the brush tool on the empty top layer (e.g., Fig. 8d). The users are encouraged to use multiple brush colors and also create their own doodle using an additional layer. The task objective is to test if users are able to work with layers, use the brush tool effectively, and detect potential synchronous conflicts. We stopped this exercise once the users were familiar with the brush tool and working with layers; this task took (10min to 15min).

Puzzle (Task-2). We provide the users with a set of disarranged pieces of a test image (Fig. 8b). Each piece is represented in the form of a single layer. The task is to rearrange these layers using rotation and translation in order to solve the puzzle (Fig. 8e). The task objective is to test if users are able to use layer transformation tools effectively. We also provide the puzzling image as a guide. On an average, it took between (15min and 20min) to complete.

Collage Creation (Task-3). Given a set of images, i.e., one background image and various foreground images with alpha

(a) Task-1 Input   (b) Task-2 Input   (c) Task-3 Input

(d) Task-1 Output   (e) Task-2 Output   (f) Task-3 Output

Fig. 8 Exemplary results obtained with our system during sessions of the post-deployment user study

matte (Fig. 8c), the users should create and layout respective layers—comprising as many foreground images as they like—in order to create a collage collaboratively. In addition thereto, they are encouraged to apply different image effects (using VCAs such as contrast enhancement, pixelation, chroma-zoom, or chromatic aberration) to each layer. The task objective is to test if users are able to reuse their learning from the previous tasks and also test familiarity with blending and layer modification via VCAs. The time for this task was limited to 15min.

5.1.3 Data collection and analysis

The online session of the above tasks is followed by a subjective interview (of approx. 15min) with questions focusing on performance, collaboration, and potential applications. In addition thereto, the entire online session was video recorded to analyze groups' collaborative practices and also to record their feedback. After the interview, each participant is asked to file a post-study questionnaire based on QUIS [4] and CSUQ [16] without any time constraints.

All the participants were able to perform Task-1 quite easily and were satisfied with the system performance. It indicates that even in the current state our system can be used for a collaborative coloring-book application. The above was a prime motivation to extend our system for the editing of vector images as well. For Task-2, the major difficulty was maintaining the control of a particular layer. Participants reported that the user-specific visual feedback regarding layer selection was too subtle. Thus, it happened that two participants were trying to move the same layer and faced unexpected results. However, in the subjective interview, they confirmed that such editing conflicts could have been avoided with the layer locking functionality. For Task-3, the major challenge was in terms of adding effects to layers, most of the users were not able to figure out this functionality on their own. Overall the user feedback can be summarized into the following two categories.

Collaboration. As expected, the novel collaborative aspect of our system was appreciated by most of our participants (Fig. 9a). They showed a great interest in having

(a) Overall Satisfaction (b) Functional Satisfaction
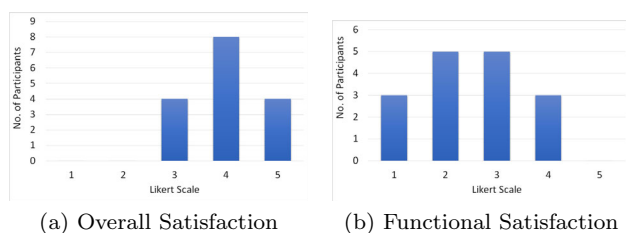
Fig. 9 The **A** overall and **b** functional satisfaction of the participants during the post-deployment user study on a Likert scale of 1 to 5, with 5 being the best

this collaborative functionality integrated into the image editing tool of their choice. Our participants from different background suggested a broader utility of our system in domains of engineering, architecture, teaching, entertainment, academia, etc., thus indicating a wide user base. However, further improvements for collaboration was suggested mainly in terms of (*i*) an integrated voice communication channel, (*ii*) hiding layers created by other team members, and (*iii*) functionality known from collaborative document editing, e.g., tagging and commenting.

Editing. Our prototype does not offer all the editing functionality generally available in a common image-editing application. Most of the participants who are familiar with such tools noticed the lack of such functionalities (Fig. 9b), e.g., an eraser tool, a flood fill tool, or selective layer manipulation (applying VCAs only on a selected region of a layer). However, the integration of collaborative versions of these tools is supported by our architecture.

To answer the initial questions as part of the post-deployment user study: (*i*) the visualization metaphors, to avoid editing conflicts, were not intuitive in the beginning but were easy to use after guidance, (*ii*) the users understood the general structure of the GUI, and (*iii*) users were quite satisfied with our prototype, especially with respect to its collaborative nature. Due to popularity of Task-1 and Task-3 among participants, we extended our system to support editing of vector images.

5.2 Editing of vector images

An SVG image with simple geometric objects generally has a smaller file size compared to a raster image and is easily editable. However, with growing geometric complexity, there is a significant increase in required memory leading to noticeable decline in performance. For multi-layer collages of raster and vector graphics, the above becomes an important factor that needs to be considered. In the following, we evaluate our system with respect to the trade-off between vector image size and corresponding performance.

As an example image, we used a *Mandala* SVG of moderate geometric complexity comprising 122 path-objects and a total size of 330 kB(Fig. 10a). To monitor system performance with increase in memory footprint, we create multiple instances of the above and measure corresponding *render* and *update* time. Figure 11a shows that with an increasing number of path-objects (and thereby size), the initial rendering time increases linearly. Following this, the delay between the application of the fill tool and the displayed visual feedback increases similarly (Fig. 11b). Thus, the performance of our system degrades linearly with the increase in memory size. As a standalone large file, we also tested editing of Fig. 10b (number of objects: 47000, size: 23 MB). The initial rendering requires 7s, and the direct manipulation has a delay of

(a) Mandala   (b) Artistic SVG
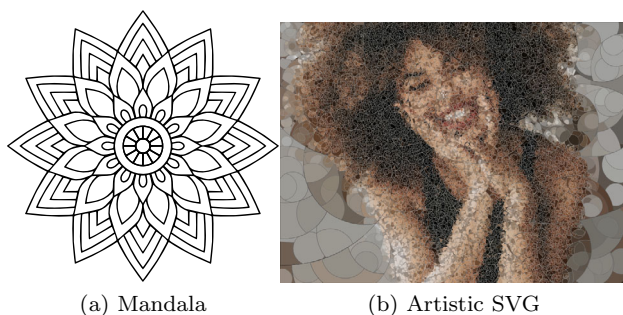
Fig. 10 Vector images used for performance evaluation

(a) Initial render time   (b) Update time
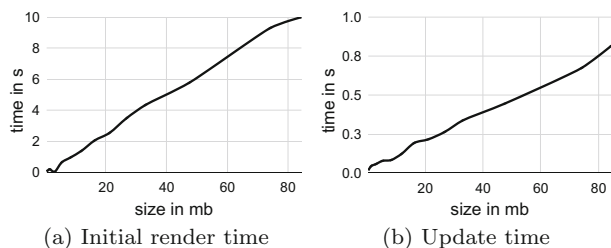
Fig. 11 Performance evaluation in regards of their size in MByte and time in second

0.8s. We observed similar rendering time and delay in other web-based vector graphics editing applications for such a large file size.

5.3 Performance evaluation

For a quantitative evaluation of our prototypical implementation, we track performance indicators for data throughput and update latency during the post-deployment study. In addition thereto, we measure rendering performance for projects of different complexity.

5.3.1 Data throughput & update latency

Data Throughput. Since rendering is client-based, and messages are sent using a WebSocket connection, only a small amount of data needs to be exchanged between the server and clients. Table 2 shows an overview of the exchanged data during the respective user-study tasks by the example of a single client. In general, the data throughput varies depending on the type of use, e.g., doodling or image manipulation. It can be observed that the most significant data and the majority of the overall data are the continuously exchange of the pointer positions. Exchanging the position with additional information, i.e., click events, supports further interaction between users and thus contributes to the real-time collaboration aspect. Despite this and regardless of the type of use, the general data throughput was overall small. This demon-

strates that the chosen architecture avoid problems arising from excessive data transmission.

Update Latency. In general, the update latency depends on the number of simultaneously active users. Due to the WebSocket connection between the server and clients, the exchanged messages during a session are rather small. Assuming a stable connection, messages can therefore be exchanged quickly and update latency is barely noticeable.

5.3.2 Rendering performance

The rendering performance of our client application depends on various factors, mostly image resolution, client hardware, and browser type. For the conducted performance tests, we use an image resolution of 1920×1080 pixels and the application is shown full-screen. We choose Google Chrome to document the performance tests, since it was the most used browser in the user-study. For testing, an average computer setup was used (i7-4790K, 16GB Random Access Memory (RAM), GeForce GTX 960 2GB Video Random Access Memory (VRAM), Windows 10). We prepared projects of different complexity, i.e., number of layers, VCA type, and amount of strokes on each layer.

Table 3 shows the averaged results of 20 consecutive rendering passes. The initial loading time of a session increases with the amount of layers and the complexity of the VCAs used. This behavior is similar to updating VCA parameter values. However, such state changes are only submitted after a user presses an "apply" button; thus, rendering a VCA does not occur that often. Also, the number of simultaneously applied VCAs is usually low.

Further, strokes drawn on the canvas do not noticeably affect the initial loading time of the canvas. Nonetheless, the performance decreases as the number of strokes increases. Since strokes are represented as individual objects rendered subsequently, this significantly impacts the rendering performance, especially if many users are drawing simultaneously. Thus, the update delays are most noticeable on mobile devices. Currently, this constitutes a bottleneck that can be approached by optimization techniques such as batching. By maintaining a state management, undo-&-redo actions for strokes can still be implemented with a batched version. During a session, the canvas only updates areas that are affected by an update and which are visible in the viewport. Hence, the resulting update time is relative to the layer and screen resolution as well.

6 Conclusions

In this work, we designed and evaluated a web-based system for real-time collaborative editing of raster and vector images. To the best of our knowledge, ours is the first sys-

Table 2 Comparison of client-side data throughput (in KByte) with respect to the event size, the count of events, and user tasks. Numbers represent the average amount of data transferred (received and sent) in 5 minutes

Event	Size	Task-1		Task-2		Task-3	
		Count	Total	Count	Total	Count	Total
Receive			3874.9		3699.6		4847.0
Pointer Position	0.22	15255	3293.00	17414	3271.00	12168	2410.00
Focus Update	0.09	15	1.45	196	18.50	36	3.40
New Stroke	2.58	277	579.00	–	–	105	235.50
Update Layer	0.11	5	0.50	312	37.90	73	10.40
Upload Image	323.00	–	–	–	–	6	1918.00
Update VCA	0.12	–	–	–	–	21	2.50
Send			2535.86		2139.43		3383.26
Pointer Position	0.18	12961	2359.00	12454	2118.00	11208	1969.00
Focus Update	0.05	5	0.27	167	8.57	41	2.00
New Stroke	2.58	94	176.00	–	–	37	77.80
Update Layer	0.11	1	0.09	111	12.40	27	3.48
Upload Image	323.00	–	–	–	–	2	1329.00
Update VCA	0.12	–	–	–	–	7	0.9

Table 3 Rendering time (in milliseconds) of projects with different complexity (i.e., number of layers, number of strokes in each layer, and VCAs) for estimation of run-time behavior

	Number of Strokes			VCA	
	None	5	30	Contrast Enhancement	Chromatic Aberration
Single Layer					
Initial Display	30	40	60	570	1100
VCA Update	–	–	–	400	850
Multiple Layers (5)					
Initial Display	55	85	130	2100	4900
VCA Update	–	–	–	1100	2800

tem that provides such a wide variety of image-edits in a collaborative fashion. In order to better understand the needs for such a system, we conducted a preliminary user study. Our prototype leverages the power of WebGL for interactive browser-based rendering, while synchronization is maintained via WebSocket connections. Our interface re-uses and extends GUI concepts from common image-editing applications. The post-deployment user study indicates a substantial demand for such a system.

As part of future work, we would like to address the existing limitations in terms of synchronization and access conflicts. To achieve the above, we aim to develop pixel-level tagging and commenting functionality in form of speech bubbles. Further, an efficient version-history maintenance for document updates using a real-time database would be an interesting idea in this regard. Additionally, we plan to enable *optimistic updates*, which makes changes locally for a client and in case of no successful response from server roll backs

the changes, thereby preventing potential delays for users with poor internet connection.

References

1. Aggie io. https://aggie.io/. Accessed: 2020-10-21
2. Bath, U., Shekhar, S., Döllner, J., Trapp, M.: Colier: Collaborative editing of raster images. International Conference on Cyberworlds (CW) , 33–40 (2021). https://doi.org/10.1109/CW52790.2021.00013
3. Calabrese, C., Salvati, G., Tarini, M., Pellacini, F.: Csculpt: a system for collaborative sculpting. ACM Trans. Graph. (2016). https://doi.org/10.1145/2897824.2925956
4. Chin, J.P., Diehl, V.A., Norman, K.L.: Development of an instrument measuring user satisfaction of the human-computer interface. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '88, p. 213-218. Association for Computing Machinery, New York, NY, USA (1988). https://doi.org/10.1145/57167.57203
5. Draw chat. https://draw.chat/. Accessed: 2020-10-21
6. Dürschmid, T., Söchting, M., Semmo, A., Trapp, M., Döllner, J.: Prosumerfx: Mobile design of image stylization components. In: SIGGRAPH Asia 2017 Mobile Graphics & Interactive Applications, SA '17. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3132787.3139208
7. Edwards, W.K.: Flexible conflict detection and management in collaborative applications. In: Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology, UIST '97, p. 139-148. Association for Computing Machinery, New York, NY, USA (1997)
8. Ellis, C.A., Gibbs, S.J.: Concurrency control in groupware systems. In: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, SIGMOD '89, p. 399-407. Association for Computing Machinery, New York, NY, USA (1989)
9. Gao, L., Gao, D., Xiong, N., Lee, C.: Cowebdraw: a real-time collaborative graphical editing system supporting multi-clients based on html5. Multimed. Tools Appl. 77(4), 5067–5082 (2018). https://doi.org/10.1007/s11042-017-5242-4
10. Google: Google documents. https://docs.google.com (2014). Accessed: 2020-10-21
11. Google draw. https://docs.google.com/drawings. Accessed: 2020-10-21
12. Heer, J., Agrawala, M.: Design considerations for collaborative visual analytics. Inform. Vis. 7(1), 49–62 (2008)
13. Isenberg, T.: Interactive npar: What type of tools should we create? In: Proceedings of the Joint Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering, Expressive '16, p. 89-96. Eurographics Association, Goslar, DEU (2016)
14. Juranek, L., Stastny, J., Skorpil, V., Junek, L.: Acceleration of server-side image processing by client-side pre-processing in web application environment. In: 2019 42nd International Conference on Telecommunications and Signal Processing (TSP), pp. 127–130 (2019). https://doi.org/10.1109/TSP.2019.8768889
15. Lee, B.R.: Analysis of digital art content created through collaboration. Arch. Des. Res. 30(4), 17–25 (2017). https://doi.org/10.15187/adr.2017.11.30.4.17
16. Lewis, J.R.: Ibm computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use. International Journal of Human-Computer Interaction pp. 57–78 (1995)
17. Nováková, K., Jakubal, V., Achten, H., Matejovska, D.: Collab sketch: Case study on collaborative sketching. In: Fusion - Proceedings of the 31st eCAADe Conference, pp. 213–218 (2013)
18. Photopea. https://www.photopea.com/. Accessed: 2020-10-21
19. Pixlr. https://pixlr.com/. Accessed: 2020-10-21
20. Rempt, B., Berger, C.: Open raster specification. https://www.openraster.org/. Accessed: 2020-10-21
21. Richter, M., Söchting, M., Semmo, A., Döllner, J., Trapp, M.: Service-based Processing and Provisioning of Image-Abstraction Techniques. In: Proceedings International Conference on Computer Graphics, Visualization and Computer Vision (WSCG), pp. 97–106. Computer Science Research Notes (CSRN), Plzen, Czech Republic (2018). https://doi.org/10.24132/CSRN.2018.2802.13. http://wscg.zcu.cz/WSCG2018/Short/P97-full.PDF
22. Salvati, G., Santoni, C., Tibaldo, V., Pellacini, F.: Meshhisto: collaborative modeling by sharing and retargeting editing histories. ACM Trans. Graph. (2015). https://doi.org/10.1145/2816795.2818110
23. Sun, C., Chen, D.: Consistency maintenance in real-time collaborative graphics editing systems. ACM Trans. Comput.-Hum. Interact 9(1), 1–41 (2002)
24. Viggiato, M., Terra, R., Rocha, H., Valente, M.T., Figueiredo, E.: Microservices in practice: a survey study. CoRR (2018). arXiv:1808.04836
25. Wegen, O., Trapp, M., Döllner, J., Pasewaldt, S.: Performance Evaluation and Comparison of Service-based Image Processing based on Software Rendering. In: Proceedings International Conference on Computer Graphics, Visualization and Computer Vision (WSCG), pp. 127–136. Computer Science Research Notes (CSRN), Plzen, Czech Republic (2019). https://doi.org/10.24132/csrn.2019.2901.1.15
26. Wu, C., Li, L., Peng, C., Wu, Y., Xiong, N., Lee, C.: Design and analysis of an effective graphics collaborative editing system. EURASIP J. Image Video Process. 2019(1), 50 (2019). https://doi.org/10.1186/s13640-019-0427-6
27. Zhai, J., Li, Q., Li, X., Wenyin, L.: A cooperative image editing tool over mobile phones. In: Proceedings of the 11th International Multimedia Modelling Conference, MMM '05, p. 264-270. IEEE Computer Society, USA (2005)

Ulrike Bath studied B.Sc. mathematics and computer science at the Freie Universität Berlin (2014–2019) and is currently studying M.Sc. IT-Systems Engineering (ITSE) at Hasso Plattner Institute, University of Potsdam.

Sumit Shekhar studied M.Sc. visual computing at Saarland University (2015–2017). Since 2018, he is pursuing his Ph.D. at Hasso Plattner Institute, University of Potsdam, under the supervision of Prof. Dr. Jürgen Döllner. His research focuses on creating efficient image and video processing algorithms based on their intrinsic attributes.

Julian Egbert is studying B.Sc. IT-Systems Engineering at Hasso Plattner Institute, University of Potsdam.

Julian Schmidt is studying B.Sc. IT-Systems Engineering at Hasso Plattner Institute, University of Potsdam.

Amir Semmo is the head of Research and Development at Digital Masterpieces GmbH and a post-doctoral researcher with the Computer Graphics Systems group of the Hasso Plattner Institute at the University of Potsdam, Germany. There he received his doctoral degree in 2016 on the topic of non-photorealistic rendering for 3D geospatial data. His principle research topics include non-photorealistic rendering, image and video abstraction, computational aesthetics and GPU computing. He is particularly interested in expressive rendering on mobile devices under the umbrella of interactive casual creativity and processing of multi-dimensional image data.

Jürgen Döllner obtained his doctorate in computer science at the University of Münster (1996) on modeling and rendering in computer graphics and habilitated after stays abroad. Since 2001, he is professor for analysis, planning and construction of complex systems at the Hasso Plattner Institute of the University of Potsdam. His work focuses on visual computing, especially in the areas of geospatial analytics, software analytics and video analytics. His visual analytics group has so far given rise to a number of software technology start-ups.

Matthias Trapp studied computer science at the University of Potsdam and the Hasso Plattner Institute, Germany (2000–2007), where he received his Ph.D. in computer science (2013). During his post-doctoral studies, he was heading the junior research group on "4D-nD Geovisualization" (2012–2017). Since 2017, he is a senior researcher at the Hasso Plattner Institute. His major research areas are computer graphics, image and video processing, geovisualization, software visualization and information visualization with a focus on GPU-based techniques.