A Framework Enabling Real-time Multi-user Collaborative Workflow in 3D Digital Content Creation Software

Swann Martinez
Paris 8 University
INREV research team
2 rue de la liberte
France 93526, Saint-Denis
R&D engineer at CUBE CREATIVE
swann.martinez@protonmail.com

Chu-Yin Chen
Paris 8 University
INREV research team
2 rue de la liberte
France 93526, Saint-Denis
chen.chuyin@mx.nthu.edu.tw

ABSTRACT

This paper reports on a graph-based approach to enable real-time update of 3D shared elements over the network between multiple artists working simultaneously on the same 3D scene. It presents an experimental framework for exploring real-time collaboration in Digital Content Creation such as animation. The framework combines a push-pull network architecture and data translation protocol with hybrid command/data replication mechanisms. This enables the synchronization of object components and hierarchy between multiple instances of the same Digital Content Creation application in a non-destructive way. Conflicts between users are prevented with a strong right management system. We demonstrate the interest of such an approach by means of an application example in Blender and discuss collaborative experimental sessions outcomes over a Local Area Network and through the Internet.

Keywords

Real-time collaboration, 3D creation workflow, Multi-user, Graphical human computer interfaces, 3D Scene Dependency Graph Replication

1 INTRODUCTION

An animation movie is an idea shaped throughout the production process by many people. A typical 3D animation pipeline requires a linear succession of tasks, from storyboarding to compositing through various stages; its workflow is similar to an assembly line.

Collaboration between each task of the production chain relies on the exchange of individual work files (see Fig. 1). The dataflow supports the workflow: the transition between steps requires export phases that format the file resulting from the previous task into a new file for the next task's tools.

Each production stage is handled by an almost entire studio department. It is therefore common that an artist working in department **A** does not know what another person did during a previous step in department **B**. This lack of communication (also known as the "silo effect") often has a negative impact on the final production by hindering error anticipation. A problem may go unnoticed for several successive steps before being discovered. When this occurs, the linear nature of the production pipeline requires going back to the root of the issue and redo all the subsequent steps, which is a very costly process.

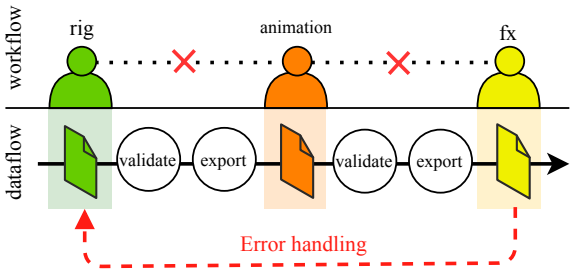


Figure 1: A traditional dataflow in animation productions

If real-time rendering engines speed up traditional linear productions by drastically reducing iteration time within the different stages, they could also become a new communication tool within animation production teams by bringing 3D designers to collaborate with each other across a real-time multi-user interface.

Real-time collaboration is widely used in other fields. In software development, it appears in peer-programming solutions such as Teletype [Git17] to

facilitate knowledge transfer. In civil engineering, Autodesk BIM[1] is an interdisciplinary collaboration hub where each actor of the construction (architects, engineers and contractors) sees in real time the impact of their work on the others.

Bringing real-time collaboration to the heart of creative applications in animation is a complex problem. From a technical point of view, the 3D scenes of a production are made of a wide variety of large software-dependant creation data (meshes, materials, rigs, textures,...). From a human point of view, the vertical communication flow and the segmented validation process do not allow for an easy multi-user collaboration.

The experimental work described in this paper results from a research-creation approach. Our framework emerged from many collaborative real-time creation sessions conducted in industrial and academic context thanks to its initial Blender integration (see section 4).

In the course of these experiments, we identified several requirements for the development of an efficient multi-user collaboration within a 3D authoring software (that will be hereafter also referenced by the name of Digital Content Creation Software or DCC):

- User interface independence: do not interfere in the interfaces of the 3D software, let the users use it as it is designed to work.
- Asynchronous editing of the 3D scene: allow different users to edit different parts and aspects of the scene simultaneously and without any friction.
- Error independence: An error caused locally by one user should not impact other users.
- Flexibility of integration: allow the DCC program to drive the replication pipeline.
- Non-destructive dataflow: allow the DCC program to specify the data types it wants to replicate.
- Adaptive workflow: adapt the behavior of the collaboration to the needs of the artists (e.g. be able to disable the replication of certain data).
- Dynamic loading: allow artists to join and leave a collaborative creation session at any time.

To address these needs, we propose a framework that brings real-time collaboration to the heart of 3D authoring software. We articulate the critical notions of dataflow and workflow at the core of the framework thanks to a data definition interface, along with a set of collaboration-aware functions allowing to model the collaborative workflow to the artists' needs (layout, lighting, modeling tasks, etc.).

Our method uses an acyclic dependency graph to store and structure the creation data for replication across the corporate Local Area Network or through the Internet. Each node contains serialized blocks of data, also called datablocks, describing any element of the 3D Scene that is stored and used by the 3D DCC software (for example: a shader, a mesh, a particle system, etc.). A translation protocol for replicated data (i.e. RDP) enables the definition of node types to match the data structures of the 3D software API.

As soon as a datablock type has been defined in the RDP, the framework will replicate instances of this given datablock type across all the connected clients. To avoid access conflicts, each node is subject to a strict modification rights management policy. This node-based approach provides an adaptive level of granularity to address the complexity of data that is specific to 3D scenes used in animation.

The user and the rights management system enable the storage of user-specific metadata in order to support the addition of new collaboration tools. These tools contribute to improve the artists' collaboration awareness by smoothing and optimizing the workflow of the multi-user creation experience.

2 RELATED WORK

This section focuses on previous work on the topic of real-time co-creation and explains the differences with the proposed approach.

The first attempts to design a framework offering a WYSIWIS experience (What You See Is What I See) [Mar87] in a single user software started in 1987. Dewan and Choudhary addressed this problem by proposing a high-level framework based on a semi-replicated network architecture [Dew92]. However, this approach based on a callback system does not take into account relational data, a crucial element for 3D creation where the entire scene is formed by a multitude of data blocks depending on each other.

Indeed, a large majority of 3D DCC solutions such as Blender[2] or Autodesk Maya[3] use dependency graphs to efficiently handle the updating of scene elements. By relying on a similar structure, our approach is thus particularly adapted to the specific needs of 3D DCC software.

In animation, real-time collaboration solutions can be categorized as cross-DCC or mono-DCC applications. A mono-DCC approach consists in replicating information within several instances of the same application. In contrast, a cross-DCC application will propagate the collaboration between different applications (Example: a texturing application, a modeling application, etc.).

By relying on the *Universal Scene Description* created by Pixar (i.e. USD) [Pix21] to manage 3D scene data, the Nvidia Omniverse solution [Nvi21] represents an

[1] https://www.autodesk.com/solutions/bim

[2] https://www.blender.org

[3] https://www.autodesk.com/products/maya

interesting cross-DCC collaborative platform. However, the standardization of 3D scene transmission formats is still limited. Some data types such as rigs are not supported. As a result, the Nvidia approach [Nvi21] is restricted to the data types supported by the USD [Pix21]. Moreover, it is complex to obtain a homogeneous interaction interface in the cross-DCC approach due to the diversity of user experiences (UX) of the solutions. To overcome these limitations, our framework focuses on a mono-DCC approach and does not impose a specific 3D data format; instead, it provides an interface for specifying the structure of the replicated creation data and their relationships to fit the application's specifics.

From an architectural point of view, the *Verse* protocol [Hni11] provides an efficient solution to the delays of 3D data transmission, but it relies on a centralized architecture. Very early in our experiments, the artists asked us for the possibility to suspend the replication of certain data on demand in order to iterate changes without impacting the other users. This requires saving updates locally, which is impossible when operations are centralized.

Another interesting approach is the Mixer add-on [Ubi20] developed for Blender by UBISOFT. Its centralized command-based architecture relies on small transfers which resulted in good responsiveness from a performance perspective. It handles the scene representation as a linear stack of commands (stored on server-side). The stack grows during the scene creation. This architecture does not provide the relational data required to prevent user editing conflicts, which is a critical need we identified in our first experiments. Since artists usually edit several aspects of an asset, the access control system must be aware of that asset's dependencies in order to lock them. Moreover, its centralized architecture does not allow the execution of local operations because it fully relies on the order of commands to rebuild the scene.

In the *MMConf* infrastructure [Cro90], Crowley and Milazzo demonstrate that a replicated architecture enables the implementation of client-specific operations when the environment is copied locally. By having a complete version of the dependency graph managed locally on each client, our framework is able to perform local operations without impacting the other connected clients.

3 PROPOSED METHOD

None of the previous approaches allowed us to explore real-time collaboration within several instances of the same 3D authoring tool in a non-destructive way and without editing conflicts.

The proposed approach is designed to be integrated in *collaboration-transparent programs* [Lau90], such as 3D authoring software that is completely unaware that it interacts with several users active on the same scene. This framework contributes to solve some of the problems listed above about multi-user experience implementation for the 3D content creation in animation thanks to the following features:

- A replicated data translation protocol: Acting as an interface for the definition of replicated datablocks.
- A session system: Managing the connection/disconnection of users at any time during the scene creation.
- An access control system: Preventing concurrent datablocks edition conflicts.
- Collaboration-aware functions: Providing functions to manage the replication pipeline behavior from the DCC application programming interface.
- Collaboration-aware interface: Providing an interface to enable the creation of collaboration coordination tools (e.g. to help users to organize themselves).

3.1 A multi-user abstraction layer for 3D content creation software

In animation, most of the 3D DCC software solutions provide a python API to extend their functionalities and integrate them in a production pipeline. Therefore, our multi-user framework has been designed as a python module to facilitate its integration.

In traditional 3D DCC, users can access many operators in order to create and modify a 3D scene. Depending on the 3D authoring software, the creation workflow will be more or less procedural. In a classic linear pipeline, a loss of information occurs during the export between tasks. This ensures to lock the artistic aspect for the next tasks. But in the context of real-time multi-user collaboration, tasks are parallelized. Thus, keeping available all the creation information in a non-destructive way is mandatory to guarantee the potential parallelism of any task within the creation. However, operators are often bound to a run-time context. Replicating an operation would require synchronizing the associated context and imposing it to other clients when the operation is being applied, resulting in potentially unexpected interface behavior and degradation of the user experience.

As a consequence, we opted for a data replication approach. This has several benefits with respect to our initial needs.

- It preserves the ability to collaborate on procedural data (e.g. a nodal network to generate geometry).
- It facilitates the implementation of a distributed architecture to apply user-local operations on data.
- It allows to extend creation data with an access control system.

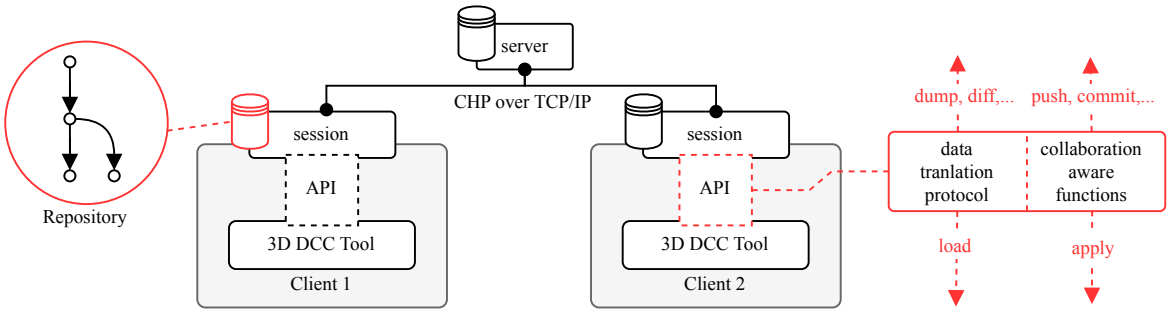


Figure 2: Overall framework architecture

In our framework, a *client* and an *application instance* are considered equivalent. Clients connect to a session stored on a replication server. The *session* is a state object over which the server has authority; it can be compared to the GameState in the video game. It stores the information of the connected clients and the status of the current creation session. The *server* is a lightweight script that can be launched on the computer of any of the clients or on a dedicated server.

3.2 Framework architecture

The framework is structured as a combination of a distributed and a centralized architecture (see Fig. 2). The distributed approach handle scene data replication while the management of commands (such as locking, connecting, etc.) is centralized to ensure server authority.

Distributed data replication

The scene data is stored in a dependency graph generated by the data translation protocol. As shown in Fig. 5, this graph is stored in a *repository* object as a key-value dictionary of entries. A key represents the unique identifier of a node and the value is the associated data.

By cloning the latter during their connections, clients get a complete local copy of this data (similar to a clone operation on Git [Jun05]). Thereafter, the changes made to it will be applied locally and then replicated. This addresses the problem of error independence. By executing certain of the collaboration-aware functions (see section 3.5) locally, errors do not affect other users.

Centralized command replication

A command consists in a set of instructions to execute on the repository (e.g. lock/unlock a node). The server which has authority for access control validates whether a command can be applied. Therefore, all commands are first sent to the server for an authorization check (e.g. does the client have the right to lock/unlock this node ?). Then they are applied on the server repository and relayed to other clients for execution. The commands are used across the different components of the framework to replicate such as:

- `Authenticate`: Login to a server.
- `Clone` : Transfer a full copy of the server repository to the local client.
- `Lock/Unlock` : Used by the right management system (see section 3.4) to acquire/release the modification right on given nodes.
- `Kick`: Remove a given user from the session.
- `UpdateUserMetadata`: Used by collaboration-aware interfaces (see in section 3.6) to update the metadata of a given user.
- `Delete`: Remove the given nodes from the repository.

Network architecture

The framework implements a modified version of the *Clustered Hashmap Protocol* [Hin11] (i.e. CHP) to exchange data across the network.

To reduce the bandwidth used by data transfers, node changes are transmitted as deltas computed during `COMMIT` (see in Section 3.5). The size of these deltas varies greatly depending on the nature of the change. E.g. moving an object will generate a small delta while updating a complex mesh will be much larger. Therefore, the granularity of the transmitted data depends directly on the modifications made by the users.

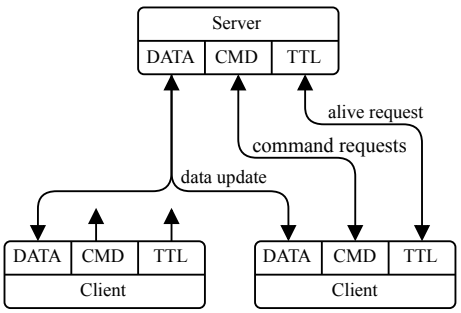


Figure 3: Network communication layer

As shown in Fig. 3, data and commands are transmitted through two different sockets. This enables to handle them asynchronously. By separating these two kinds of transactions at server level (see Fig. 4), we guarantee the responsiveness of the command-based right system even when large delta are transferred.

Each channel is based on TCP to limit packet loss. The TTL socket mechanism allows to measure the ping response and the latency to determine if the clients are online. On client side, this regular heartbeat mechanism is handled in a separate process, so that it will be less impacted by the heavy computations typical of 3D creation tools (rendering, etc.). Same strategy is applied server-side.

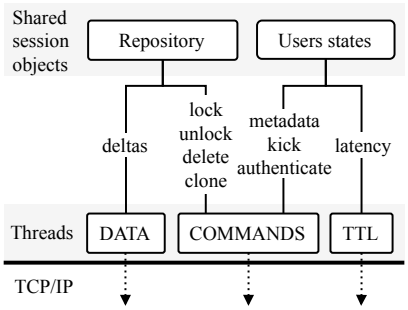


Figure 4: Server internal architecture with asynchronous network frame reception.

3.3 Replicated data definition protocol

The Replicated Data definition Protocol (abbreviated as RDP) represents the keystone between the DCC program and the framework. It acts as a translation dictionary to transport data between the two (see Fig. 5).

To enable the support of a given type of datablocks, it is necessary to define an implementation of the abstract class `ReplicatedDatablock` offering the following methods:

- `dump`: Translates the target object into a dictionary using standard types.
- `load`: Loads the dictionary data into the DCC datablock.
- `construct`: Instantiates an empty datablock.
- `resolve`: Search for an existing instance of a specific datablock.
- `resolve_dependencies`: Returns the dependencies of a given datablock (e.g. textures, meshes, etc.).

The methods above are used by the collaboration functions (see section 3.5) across the replication pipeline to exchange data between the local repository and the DCC datablocks.

The definition of these implementations takes place only once. This occurs during the integration phase of the framework into the DCC software. As they will deeply interact with the 3D authoring software, a strong knowledge of the program's python API is required. In our application example (see section 4), we developed those implementations as sub-module of the add-on responsible for the framework integration. Once defined, these implementations are stored in a key-value dictionary (see Fig. 5). The key is the datablock type and the value is the corresponding implementation. This dictionary is then transmitted to the framework during the session initial connection. In this way, the framework is able to interact with the DCC data to replicate them.

The implementation dictionary will be used all along the session by the framework to instantiate the defined implementations. These instances represent the different nodes of the replication graph stored in the repository.

3.4 Concurrency access control

Initially, as the framework had no access control system and several users were able to modify the same datablock, which naturally led to editing conflicts. To address this, we limited the modification of a datablock to a single artist by means of the right management system. The access control system manages the ownership of each node of the replication graph. It will allow or refuse/disable the modification of some nodes of the latter according to their owner. The `change_owner` function allows to change the rights on a node and optionally its dependencies. When a node belongs to a user, only this one has the ability to modify it.

By default, the right management policy assumes that a node belongs to what we call 'the common right'. That is, when a user unlocks a node, it yields its ownership to the common right. Only nodes belonging in the common right can be locked by a user. This access control system ensures frictionless collaboration by allowing users to work asynchronously on different datablocks of the 3D scene.

3.5 Collaboration-Aware Functions

The collaboration-aware functions (i.e. CA-function) are the core of the framework's replication pipeline. They are exposed to the DCC program to let it control and adapt the rate of updates to its needs. Any of them can be called automatically (by the program) or manually (by the user). The Fig. 6 shows the place of these functions in the replication chain. We took inspiration from Git [Jun05] to develop these mechanisms of collaborative work provisioning.

As shown on Fig. 5, the CA-function `ADD` first adds the targeted datablock to the local repository by instantiating a new node corresponding to its type (with the RDP presented in 3.3). This is the entry point of the replication pipeline. When a datablock is added to the repository, it is considered as being tracked. During a session, `ADD` must be called to register each new datablocks.

Then, the CA-function `COMMIT` can be executed on any node. Firstly, it will check the state of the datablock's dependencies to ensure published data integrity. For each datablock returned by

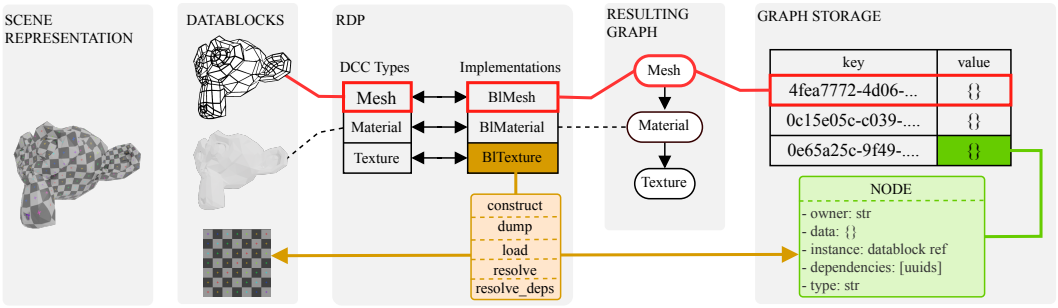


Figure 5: Data translation with the RDP

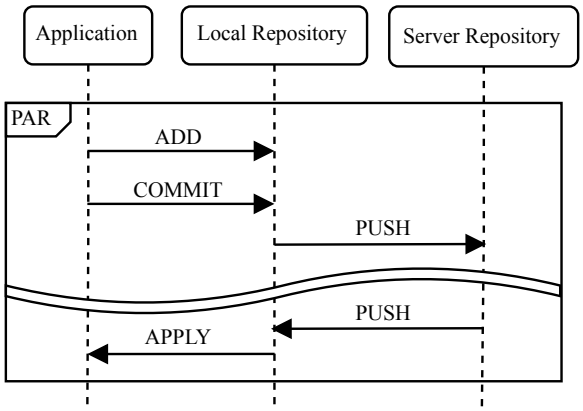


Figure 6: Data collaboration primitive pipeline

`resolve_dependencies` (see section 3.3), it will check if it is tracked and up to date. If so, the latter will be subject to an `ADD` or a `COMMIT`. Secondly it will calculate the changes and apply them to the corresponding node in the local repository. The changes consist of a differential of the data extracted (with `DUMP`, see section 3.3) from the datablock in its current state and the last committed version stored in the node in the `data` field (see Fig. 5). We use DeepDiff [Zep21] to compute the delta between the two dumped versions of the datablock.

The `PUSH` CA-function is then called to transfer local changes to the server which will apply them onto its own repository and relay them directly to connected clients.

As soon as the modifications are received by the connected clients, they can be loaded with `APPLY`. This CA-function checks the corresponding datablock existence with `resolve` (see section 3.3). If it is not resolved, it means that it is a new datablock corresponding to a new scene element. In that case, it will be instantiated using `construct` (see section 3.3). Once a datablock instance is found, the `load` method (see section 3.3) of the node implementation will load its updated data into it.

The default behavior is to `COMMIT` then `PUSH` as soon as a change occurs in the scene in order to ensure real time updates. In certain situations (e.g. working on large volumes of geometry) it is essential to give the artist the ability to temporarily stop sending data to avoid impacting the performance of other clients. By separating the replication steps into functions, we address the problem of disabling outgoing updates and the need for an adaptive workflow.

3.6 Collaboration-Aware interfaces

During 3D scene editing, the user interactively edits the assets in a spatial and temporal context. If we extrapolate this into a multi-user environment, it is important for each user to be aware of who is working on what element, where and when through collaborative user interfaces.

Key	Value
`view_matrix`	`[[0.0, 0.9, 0.0, 0.0], [-0.4, 0.0, 0.9, 0.0], [0.9, 0.1, 0.4,-5.0], [0.0, 0.0, 0.0, 1.0]]`
`color`	`[0.0, 0.3, 0.2, 1.0]`
`frame_current`	`91`
`scene_current`	`main_stage`
`selected_objects`	`['object_1', 'object_2']`

Table 1: A sample of the user metadata dictionary used in the Blender framework integration

The `user_states` object of the framework is intended to support such interfaces. More concretely, it is a dictionary of metadata (e.g. in Tab. 1) specific to each user stored and replicated across all clients. The framework provides two functions to interact with this object:

- `update_user_metadata()` allows a client to update one or more of its metadata fields across all connected clients.
- `get_users_states()` retrieve the metadata dictionnary of all the connected clients (including the local one).

To maintain collaboration awareness, it was crucial to support a high metadata refresh rate. E.g. the temporal snapping operator required a minimal 30 Hz refresh rate on the `frame_current` field. To do so, we limited the size of the updates to the strict minimum. To do so, when a client request to update a data field with `get_metadata`, only this latter

will be relayed through the network encapsulated in an `UpdateUserMetadata` command (see in Sec. 3.2) that will patch all connected client's `user_states` object.

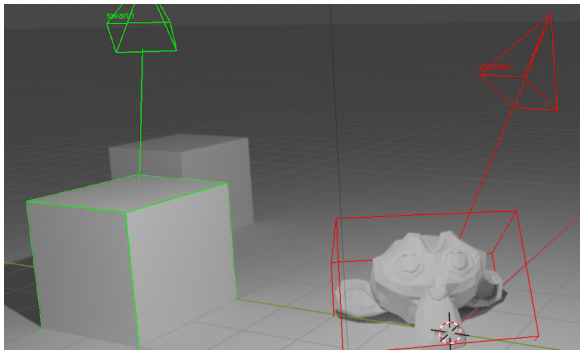


Figure 7: Collaboration aware interface in Blender viewport

For instance, the implementation of the framework in Blender relies on these interfaces to store the user's point of view (see the `view_matrix` field in Tab. 1) in order to draw their frustums in the viewport of the other connected clients (see Fig. 7). The same concept is applied to represent the active selection (see the `selected_objects` field in Tab. 1) bounding box of each user. These two user interface elements make connected artists completely aware in real time of where each other are and what they are working on.

To allow artists to find each other in space and time, two operators have been implemented in Blender. They rely on the metadata made available through a call to `get_user_states` (respectively the `view_matrix` and `frame_current` fields in Tab. 1).

4 APPLICATION EXAMPLE

As an open-source Swiss army knife for 3D content creation, Blender can handle all stages of film production.

The animation studio CUBE CREATIVE [Cub21] has made it its main DCC since 2019, from asset creation (modeling, texturing, shading and rigging) to rendering and animation.

A key advantage for real-time collaboration is its unified shader system between its real-time rendering engine (EEVEE) and its path tracing engine (Cycles), which allows for high-fidelity pre-visualization of material rendering in real-time in the viewport. Applied to the multi-user context, this provides the possibility for all users to visualize in real-time the visual changes of the scene in EEVEE while keeping a fidelity close to the final rendering in CYCLES.

We integrated our framework into Blender as an open-source add-on called *Multi-User* [Mar19]. This made possible the study of the impact of real-time collaboration in two different environments:

- An industrial one: In the studio CUBE CREATIVE (where the framework was developed) with supervisors and artists teams specialized in animated series.
- An Academic one: For teaching computer graphics at university Paris 8.

The collaboration functions described in 3.5 were placed in the callbacks provided by Blender (like: `depsgraph_update_post`). Thus, Blender directly notifies the framework of any updates of the scene and the framework publish these changes to all the clients.

5 ANALYSES AND OBSERVATIONS

From July 2019 to January 2021 we conducted a set of experimental sessions at CUBE CREATIVE using our Blender integration of the framework (the Multi-User add-on). The real-time collaboration was applied to concrete use cases reflecting the reality of a production through different exercises:

- Scene re-creation: Create scenes (any aspect) from scratch, while being guided by a 2D reference image.
- Background concept: Create scenery from already existing assets issued from a production.

As mentioned in the section 1, the production of a film linearly passes through many stages. Each step is isolated and validated one by one. Within the production, the real time collaboration can be conceived within one or several production stages. These two configurations will lead to two very different team compositions. We will refer to a multi-disciplinary team for the collaboration invoking multiple production stages, whereas for performing collectively a single kind of task, we will refer to a specialized team.

Both types of exercises were designed to test these two team configurations. For the background concept exercise, the teams are specialized. But, the scene re-creation requires several professions working together because it encompasses all aspects of creation.

These two classes of exercises allowed us to evaluate our approach in terms of efficiency and quality.

5.1 Collaboration efficiency

To study the impact of real-time collaboration on work execution efficiency, we conducted a series of scene re-creation sessions based on references images chosen for their diversity.

Each of these scenes was created twice on Blender: one version was created by a single person executing sequentially the different tasks, and the other version was created by a team collaborating in real time with multi-user add-on through the internet. In both cases, the artists had to create the following aspects:

- 3D models
- shading
- layout
- fx
- lighting
- compositing
- rendering

Despite being an abstract notion, the qualifications of the artist is a very important factor regarding the results. We can distinguish three levels of experience in animation teams: junior, mid and senior artists. In the single-user creation experiment, the artists had a mid-experience, such that they can handle all aspects of the content production. In the multi-user part of the experience, the collaborative team was mixed equally with mid and junior artists.

Scene			SU	MU	
name	*tris*	*Oc*	*t*	*t*	*Ts*
Campsite	3430	161	63	45	3
Paper Summit	22215	58	90	60	5
All seing monolith	59000	146	177	90	4
Xbox clubs image	16112	726	238	175	4
Abstract city	2267256	139	472	295	5

Table 2: Scenes re-creation sessions time (t) in minutes in single-user(SU) and multi-user(MU) configuration with corresponding team size(Ts), Total triangles($tris$) and objects count (Oc)

Tab. 2 shows how real-time collaboration can speed up the scene creation time. This gain varies according to the complexity of the scene and the size of the teams. According to our data, teams of 4 artists obtain the highest efficiency rate on complex scenes. Although encouraging, the method's efficiency could be greatly improved with an optimisation of the work distribution before the session. During the re-creation sessions, we didn't impose that, and as a result, we found the artists, by habit, tended to work individually on the assets. Thus, we observed the main factor limiting the parallelization of work was organizational. The goal would be to overlap the creation of multiple aspects of the same asset to improve the collaboration workflow parallelism. For example, one artist can go-on building an asset while another could place it in the scene.

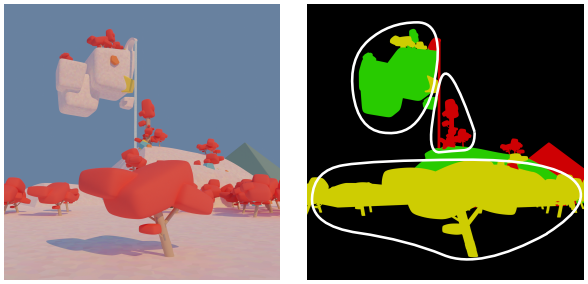
Beyond efficiency, the re-creation sessions highlighted other benefits of the real-time co-creation. Although working on individual assets, the artists were naturally led to simultaneously work on several aspects of the scene. E.g. it was common for one artist to start the lighting while other are modeling and laying out elements. As a result, they were aware in real-time of their impact on the work of others which greatly improved error handling as opposed to a linear industry pipeline.

5.2 Qualitative observations

While the analysis of the re-creation sessions has shown the efficiency of our framework for collaborative work in a concrete context (guided by references), we also questioned the ability of such a workflow in an abstract context such as in pre-production where new designs have to be created.

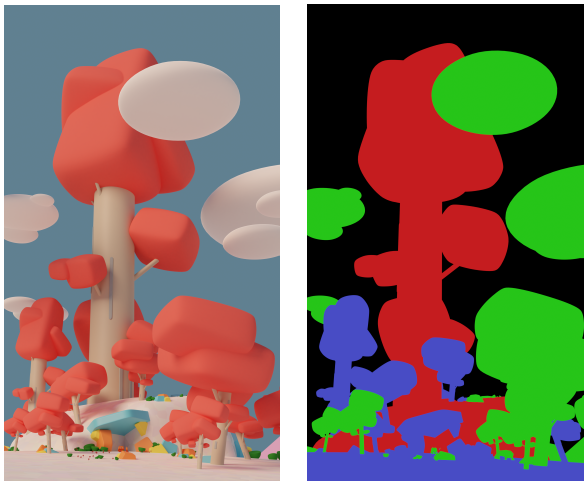
During the collaborative background concept sessions, the teams were composed of three to four members ranging from mid to senior level.

In order to be as close as possible to real life conditions, we used a set of existing assets coming from the Tangranimo series currently in production at CUBE CREATIVE. The exercise focuses on a collaborative layout to design new sets based on an existing visual style.



(a) (b)

Figure 8: Background concept session 1 result, including (a) the scene render and the corresponding (b) user work distribution by color.



(a) (b)

Figure 9: Background concept session 2 result, including (a) the scene render and the corresponding (b) user work distribution by color.

The spatial distribution of work is a particularly interesting criterion to study in this kind of improvisational exercise. It provides a visual indication of the collaboration quality: this can be measured by coloring the space regions occupied by objects according to the user

who worked on them. When the collaboration is not fluid, we observe very distinct islands that do not intersect as well as a dominance of the occupation rate of certain colors.

user	occupancy
Session 1	
yellow	3.9
green	1.9
red	0.8
Session 2	
green	6.2
blue	3.4

Table 3: Users work occupancy percentage on the background concept session results.

This was the case of the first session which was conducted without any preparatory phase. In the user's work distribution (see Fig. 8b), we observe very distinct islands that do not intersect, as well as a dominance of yellow on the spatial occupancy rate (3.9 % for the yellow user against 1.9 % for the green and 0.8 % for the red, see Tab. 3). It means that the artists were working in a spatially isolated way, revealing that the collaboration was not smooth, as if they were shy of interacting with each other. As a direct consequence, the resulting scene shown in Fig. 8a lacks of coherence.

In the second experimental session we introduced a 5-minute briefing period at the beginning of the session dedicated to settle the creation of a common foundation (in red in Fig. 9b). As shown in the user work distribution in fig. 9b, the common conception of the space resulted in a much more uniform distribution of work. The resulting scene (see Fig. 9a) is coherent.

When errors occurred (e.g., an object casting an unintended shadow), we observed that the real-time nature of the collaboration allowed the participants to notice them and instantly fix them. In a traditional production pipeline, this process would have been much more time-consuming and tedious (as shown in fig. 1). Thus, using the framework permitted to improve considerably the communication between artists, greatly increasing the anticipation of errors. Furthermore, the artists started talking to each other during the creation, which was not possible before because of the "silo effect". Adding this social dimension to the creative process has led artists to evaluate and re-calibrate their work according to the scene being created. This natural review process is intrinsic to the global vision of the project given by the real-time nature of the collaboration. It allows the artists to communicate about their practices and thus generates a natural transmission of knowledge between the different levels of experience.

Academic application

Although we mainly addressed the industrial application of this work, it turned out that it would add a new dimension to the teaching of computer graphics. For example, we used the Multi-User add-on during 3D modeling courses held at the department Art and Technology of the Image of University Paris 8 in 2020. This led the teacher to share the same virtual space as the students. By interacting with their practical work in real time, the teacher is able to react naturally and quickly to the questions and difficulties of each student. From an educational point of view, our work may have a relevant application in the academic world.

6 CONCLUSION AND FUTURE WORK

In this paper, we presented an experimental framework to explore and question the contributions of real-time collaborative work in animation. Based on a mono-DCC approach, our Replication Data Protocol (RDP, see section 3.3) supports a non-destructive dataflow thus adapting collaborative possibilities to the specifics of 3D authoring software. With the addition of collaboration-aware functions, we adapted the collaborative workflow to the constraints of the creation software and the needs of the artists.

We evaluated our framework on the efficiency and quality of the collaboration it supports. For this purpose we conducted experimental sessions based on its integration in Blender. It turned out that in addition to speeding up the process of creating 3D scenes, the presented work also increases considerably the visibility of the artists on their ongoing creation.

By adding the real-time aspect within the collaboration we have moved away from the iterative nature of the work. As presented, the framework relies on the current version of the creation data. It would be interesting to consider the validation and the tracking of the artistic work in the context of a scene created in real-time by a team. Adding a versioning layer of the changes could help the production teams to track the team's work over time. This could be achieved by saving a full version (i.e. snapshots) of the repository in different ways. An automatic strategy triggered at a regular time interval meets the backup and security requirements. In contrast, a manual strategy driven by the artist from its DCC software would be ideal to iterate precisely on a scene aspect. The latter would be used to perform validation reviews based on a given state of the work. Currently, the framework stores the entire repository in memory, limiting the size of the supported scenes to the available memory on the client computer. A future work would be to add a disk-based cache system to achieve larger 3D scenes support. By configuring it via the data translation protocol, each type of datablocks would benefit from a caching strategy tailored to its needs.

7 ACKNOWLEDGMENTS

This work has been supported by the French National Association of Research and Technology (ANRT) through an Industrial Research and Formation Convention (CIFRE №2018/0204) established between the company CUBE CREATIVE and the INREV research team of the University Paris 8. And we would like to thank in particular Valentin Moriceau, director of the R&D department of CUBE CREATIVE for his support along the project. We also thank the Blender community for its incredible involvement in the development of the Multi-User add-on.

8 REFERENCES

[Cro90] Crowley T., Milazzo P., Baker E., Forsdick H., Tomlinson R. MMConf: An Infrastructure for Building Shared Multimedia Applications. In Proceedings of the 1990 ACM Conference on Computer-Supported Cooperative Work, 329-42. CSCW "90. New York, NY, USA: Association for Computing Machinery. 1990. https://doi.org/10.1145/99332.99365.

[Cub21] CUBE CREATIVE Computer Company. http://www.cube-creative.com/.

[Dew92] Prasun D., Choudhary R. A High-Level and Flexible Framework for Implementing Multi-User User-Interfaces. ACM Transactions on Information Systems 10: 345-80. 1992.

[Git17] GitHub. Teletype. https://teletype.atom.io.

[Jun05] Hamano J. Git. https://github.com/git/git.

[Hin11] Hintjens P. Clustered Hashmap Protocol. https://rfc.zeromq.org/spec/12/.

[Hni11] Hnidek J. Network Protocols for Applications of Shared Virtual Reality. Journal of WSCG, vol. 19, pp. 31-38.

[Lau90] Lauwers J.C., Lantz A.K. Collaboration Awareness in Support of Collaboration Transparency: Requirements for the next Generation of Shared Window Systems. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 303-11. CHI "90. New York, NY, USA: Association for Computing Machinery. 1990. https://doi.org/10.1145/97243.97301.

[Mar87] Stefik M., Foster G., Bobrow D. G., Kahn k., Lanning S., Suchman L. Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings. CACM 30:1, pp. 32-47. January 1987.

[Mar19] Martinez S. Blender Multi-User Add-On. 2019. https://gitlab.com/slumber/multi-user.

[Nvi21] Nvidia Omniverse platform. https://developer.nvidia.com/nvidia-omniverse-platform.

[Pix21] Pixar. Official USD documenation website. http://graphics.pixar.com/usd/docs/index.html.

[Ubi20] Ubisoft. Mixer. 2020. https://gitlab.com/ubisoft-animation-studio/mixer.

[Zep21] Zepwork. DeepDiff. https://github.com/seperman/deepdiff.