A Data-Efficient Collaborative Modelling Method using Websockets and the BlobTree for Over-the Air Networks

Herbert Grasberger[*] University of Victoria
Pourya Shirazian[†] University of Victoria
Brian Wyvill[‡] University of Victoria
Saul Greenberg[§] University of Calgary

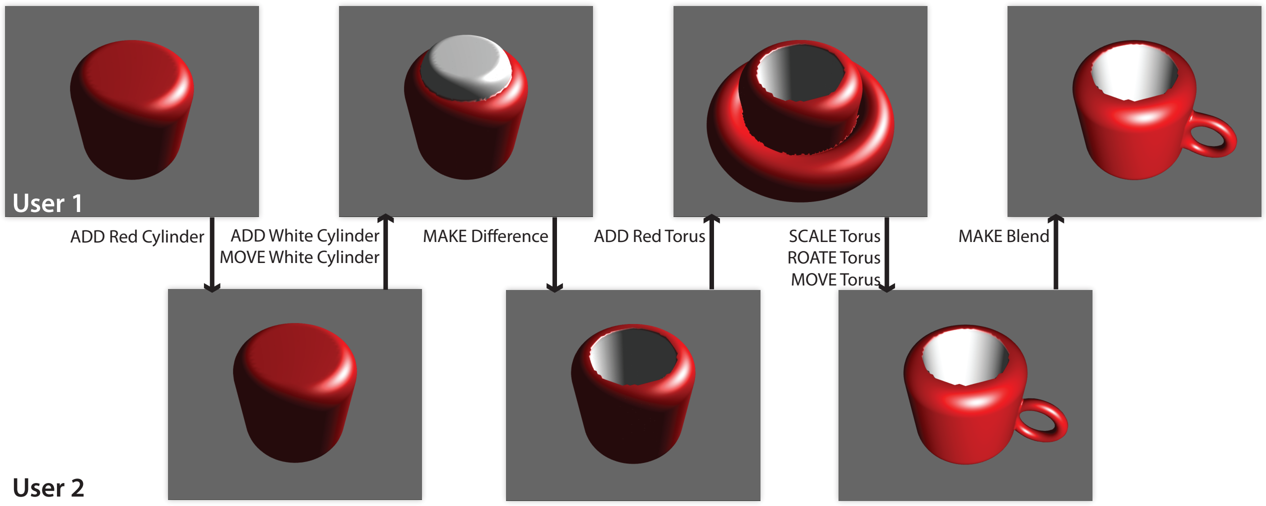


Figure 1: *An example modelling session between two users.*

Abstract

Collaborative modelling has become more important in the last few years, especially now that mobile devices show processing power to support 3D modelling in real-time. Current mobile networks, such as 3G and LTE, unfortunately are not as fast as traditional wired internet and have higher latency.

The problem with collaborative modelling using triangle meshes is that complex models are slow to synchronize and require large network resources depending on the amount of data needed to update a model. Synchronizing thousands of triangles over the network between all participating users can introduce substantial lag between the transactions, especially on over-the air networks, making fine grained and rapid updates at interactive rates hard to achieve.

In contrast the *BlobTree* is based on combining skeletal primitives and sketched-shapes using standard CSG and various blending operators. Using this methodology complex models can be encoded with a smaller memory footprint than mesh based systems, thus allowing for less traffic across a network to synchronize two or more workstations with one model. As a result fine grained and rapid updates are possible, improving the visual communication between all participating users.

CR Categories: I.3.2 [Computer Graphics]: Distributed/network graphics; I.3.5 [Computer Graphics]: Modeling packages;

Keywords: Collaborative, Distributed Systems, BlobTree, Websockets, Implicit Modelling

1 Introduction

This research is motivated by the desire to work collaboratively and share highly complex models across the network. With the introduction of touch-based tablet devices, a new way to share and model objects collaboratively using touch and sketch-based input is possible. Large models are very likely to be constructed by more than one person, particularly for product design where designers of different model-parts may be at disparate locations.
The frame of a bike for example is sketched by a designer, whereas the linkages for the suspension are created by an engineer. Additional parts are added by another designer to create a final production rendering. Since network speed can be a limiting factor in collaborative design, one of the main criteria for our system is its small memory footprint and reduced amount of necessary synchronization messages.

In the VRML/X3D strategy the majority of the 3D geometric data shared and transmitted on the network is a polygon mesh. Mesh compression approaches and progressive meshes try to reduce the amount of information transferred, sometimes by reducing the overall quality of the mesh.

In comparison our proposed system minimizes network loads by transmitting updates to the hierarchical structure known as the

[*]e-mail:grassi@cs.uvic.ca
[†]e-mail:pouryash@cs.uvic.ca
[‡]e-mail:blob@cs.uvic.ca
[§]e-mail:saul@ucalgary.ca

BlobTree [Wyvill et al. 1999], where every participant receives the most precise description of the model. Our design sends the information as typed messages with their associated parameters representing user modifications to the model. This strategy keeps the scene structures synchronized across multiple design stations. The *BlobTree* data-structure is modified by each participant using the commands and visualization of the model is performed locally on each system using available processing resources.

The *BlobTree* is both a data structure and a modelling paradigm similar to Constructive Solid Geometry, as in [Ricci 1973]. It is based on the combination of skeletal implicit primitives with boolean nodes, as well as more advanced operators to create complex shapes, including warping, filleting and blending between the nodes. The resulting *BlobTree* is a complete, and compact description of an implicit model.

Both, implicit models and the operators used in the *BlobTree* can be described using a small set of parameters, thus the *BlobTree* is a good choice for a compact data description to send across the network. The *BlobTree* can be polygonised for fast visual feedback, or if a high quality image is desired it can be rendered using a ray tracing approach. Furthermore the *BlobTree* supports rapid prototyping via sketching as described by [Schmidt and Wyvill 2005] where the basic building block of a sketch primitive can be very small in terms of memory.

The contribution of this work is a synthesis of existing techniques from different disciplines. We minimize network traffic by using a hierarchical implicit modelling system, we use a sketch based metaphor for direct manipulation of a model, and a layered serverless messaging system based on Websockets [W3C 2013] that does not require locks for synchronization. Together these improvement create a system contribution that could have impact on the way models are built in a collaborative environment as shown in Figure 1. This also exposes a big advantage in using skeletal implicit modelling, a previously little utilized approach. In real world situations where only 3G/4G networks are available over hand held devices, this method shows a big advantage since fine grained and rapid updates of the scene are possible due to the *BlobTrees* small memory footprint, enabling the system to be highly interactive. Our approach is lock free and enables 'simultaneous modifications'. When several users want to change the same feature of a model in different ways, all of them can grab this feature and modify (e.g. translate or rotate it) at the same time.

The remainder of this paper is organized as follows: related work in distributed collaborative modelling is found in section 2, the *BlobTree* and its basic operators are described in section 3. Our system of network messages is explained in section 4.1, with an in-depth discussion of each message layer in section 4.1.1, 4.1.2 and 4.1.3. Synchonization issues are discussed in section 5 and our unique user interface features are explained in section 6. The work continues with some example objects modelled using our system including a discussion on the of data transmitted in section 8. Finally, conclusion and future work is given in section 9.

2 Related work

Mouton [Mouton et al. 2011] provides an in-depth analysis of current collaborative environments, mainly targeted to handle visual data sets. They suggest that new applications should try to reduce their usage of bandwidth by using local client resources to increase an applications interactive performance. They advocate a focus of new applications towards transferring less data and calculating more information. In addition they suggest that developers of new applications should try to use given standards instead inventing their own. Our approach conforms with this idea, in that it uses low bandwidth and uses HTML Websockets for transferring the information.

One early distributed virtual environment for engineering and manufacturing was CollabCAD [Mishra et al. 1997]. In this system a mesh model is shared across the network amongst multiple designers. Previously designed models are imported for further manipulation, and detailed modifications. Concurrent access to a common design is enabled for viewing and modification.

Nishino [Nishino et al. 1999] created a collaborative modelling environment to enable the design of implicit models. Each participant in the system can login to a session server to gain access to the part of the object being designed by other participants on that server. All session servers are managed by a centralized world server, which controls access rights and updates done by all participants. To make a modification to the model each participant requests an update right which is acknowledged by the session server. Each client holding an update right sends updated parameters to all other participants connected to the same session server, then it releases the update right and saves the tree data to the session server, not allowing simultaneous input from several nodes. Our work uses the same basic idea of an implicit modelling system. We improve on Nishino's system by using the *BlobTree*, giving us a variety of primitives including sketch-based shapes, CSG and different types of blending and deformations [Sugihara et al. 2010]. We avoid problems associated with having a central session server handling update rights (e.g. when the server fails), by offering a server less system that allows multiple designers to make changes to the model simultaneously.

Many mesh-based approaches, such as the ones by *Han* [Han et al. 2003], *Ramani* [Ramani et al. 2003] and *Kim* [Kim et al. 2006], also employ client-server architectures. Meshes are transferred between the clients and the server, never in between clients. All of these approaches use different ways to control access to parts of the model, all controlled by the central server. Some approaches try to compensate for the lack of synchronization speed the mesh approach creates, by adding a mesh hierarchy or a level of detail method, such as the one presented by *Chu* [Chu et al. 2009]. One of the problems is that as the detail and therefore the complexity of a model increases, updating the mesh hierarchy, rapidly becomes the bottleneck in the system. This is in addition to the issues of client-server based approaches.

Our work is based on Websocket [W3C 2013] which is a standardized protocol to transfer messages across the internet, based on HTTP. *Marion* [Marion and Jomier 2012] uses a Websocket implementation to transfer the scientific data to and between their visualizer clients. At program startup only the data set is transferred and only once this is finished the collaboration process starts, where users can then work on the data set concurrently, however the data set cannot be changed interactively. They highlight that their Websocket implementation can achieve lower latency and a higher synchronization rate than a comparable AJAX implementation.

Distributed sketching has been a topic of long interest in the Computer Supported Cooperative Work (CSCW) and groupware community. While most reported systems are for simple 2D sketches, the human and social factors underlying distributed interaction apply equally to 3D modelling. These are perhaps best summarized by the mechanics of collaboration that cover the basic communication and coordination operations of teamwork - the small-scale actions and interactions that group members must carry out in order to collaborate within a shared workspace [Pinelle et al. 2003]. In brief:

- *Explicit communication* occurs not only through spoken and written messages, but by gestural messages, deictic refer-

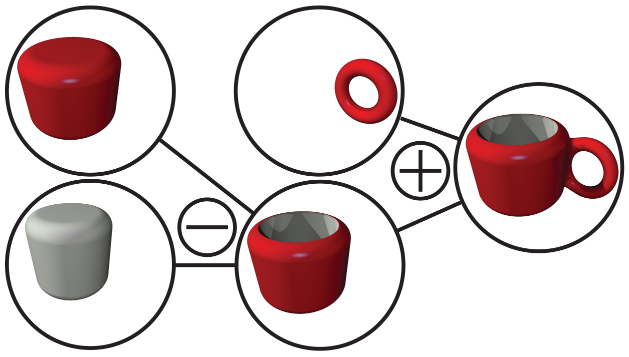


Figure 2: *An example BlobTree. Primitives are combined to form a mug.*

ences and actual actions that accompany talk (e.g., indicating, demonstrating, pointing, moving a pen to initial drawing, drawing actions).

- *Information gathering* includes fine-grained knowledge of what others are doing. This includes basic awareness (who is in the workspace, what they are doing, where they are working), feedthrough (changes to objects made by others), consequential communication (body position and location, gaze awareness).
- *Shared access* describes how people access tools and drawing objects, which covers how they reserve and obtain such resources, and how they protect their work by (for example) monitoring others' actions in an area and negotiating access.
- *Transfer* covers how people physically handoff objects to others, and how they place objects in a space so others can use them.

Technically, we require a few factors for the above to work in a real-time collaborative situation. First, people need to communicate through talk. This means a rich communication channel is necessary: in our case, we expect people to use existing systems (e.g., telephones, VOIP, video conferencing) alongside our system. Second, people need to see rapid and fine-grained updates of the 3D sketch as it evolves, including transitional states that accompany object addition, deletion, movement, transformation, and so on. If delays are excessive, or if objects just 'shift' from one state to another without displaying in-between states, people have difficulty tracking what is going on, and have problems coordinating their talk with their sketching actions. This is the main motivator for our work: by using and transmitting only a small set of parameters, fine-grained and rapid updates are possible. Third, people need to be embodied in the system in a way where others can see where they are, where they are attending, and what they are about to do. As common in most groupware, we do this through multiple cursors, implemented as arrows in 3D space and camera items implemented to show a miniature of the remote users view of the scene.

3 Representing the Model

Our choice of the underlying data-structure was motivated by the two main criteria, as stated by [Mouton et al. 2011]:

- small memory footprint and resulting little bandwidth use
- efficient use of local client resources to visualize.

The *BlobTree* fits these two criteria in that it extends existing skeletal implicit surface modelling techniques [Bloomenthal 1997]. It combines these skeletal implicit surfaces with a unified structure in which nodes can represent arbitrary blends between objects as well as Boolean operations, and warping at a local and global level. See Figure 2 for an example of the construction of a simple model. To visualize a model, whether by polygonization or ray-tracing, requires traversal of the *BlobTree* for a large number of points in space. The *BlobTree* simply returns a field-value and a gradient for each point p.

Much work has been done on improving the speed at which the *BlobTree* can be traversed to produce a triangle mesh. There is a long history of polygonization algorithms starting with the uniform voxel grid method of [Wyvill et al. 1986]. Bloomenthal published a popular implementation of the uniform grid method in [Bloomenthal 1994], which in addition, overcame ambiguities using tetrahedral decomposition. A more efficient algorithm was published in [Akkouche and Galin 2001].

The sketch-based system of [Schmidt et al. 2005a], for efficiency trades accuracy for speed by storing cache nodes in the *BlobTree* [Schmidt et al. 2005b]. For accurate visualization but in general non-interactive applications, ray tracing can be employed using interval analysis [Snyder 1992], or Lipschitz approaches such as [Kalra and Barr 1989].

3.1 Skeletal Primitives and Blend Operators

Most of the primitives used in the *BlobTree* are built from geometric skeletons, which are incorporated in many implicit modelling software packages such as BlobTree.net [de Groot 2008] or ShapeShop [Schmidt et al. 2005a]. They are ideally suited to prototype shapes of arbitrary topology [Bloomenthal 1997]. In general these works conclude that the use of skeletal primitives can lead to a simple and intuitive user modelling methodology.

The basic building block of a skeletal primitive is a skeleton S. To create a skeletal primitive the distance-field d_S of the volume encapsulating the shape has to be computed as described in [Barbier and Galin 2004]. The distance field is a volume of scalar values which is not bounded as the distance itself can be infinitely large.

By modifying d_S with a field function g, it can be bound to a finite range. Usually the function maps the distances to the range $[0, 1]$, where the field has values of 1 at the skeletons and 0 after a certain distance to the skeleton (usually at distance 1). A discussion of field function appears in [Shirley and Marschner 2009].

Skeletal implicit primitives are combined using binary operators, which are applied pair wise to field-values f, and represented by a node in the *BlobTree*, whose children are either primitives or operators themselves.

Field values are computed for the child-nodes and combined to yield a new value according to the operator type. This makes it possible to go beyond the classical Boolean operators, and define general *blend operators* that e.g. create smooth transitions between shapes. The most common operator that creates a smooth transition between several values is called the *summation blend* [Bloomenthal 1997]:

$$f_R(p) = \sum_{n \in N} f_n(p)$$

where the resulting field-value at a point p in space $f_R(p)$ is the sum of the field-values of all the objects involved.

More complex operators, such as those described in [Barthe et al. 2003] and [Barthe et al. 2004] or the blending functions that are based on R-functions [Shapiro 1994], [Pasko et al. 1995] and [Pasko and Savchenko 1994], allow for a fine control on the resulting blend shape. By using them it is possible to create complex blended shapes similar to the ones proposed for CSG in [Elber 2005].

3.2 Sketching

One big advantage of the *BlobTree* compared to other modelling approaches is its seamless integration with sketched objects, where 2D sketches are used to create 3D objects. The implementation of sketching follows the approach described by *Schmidt et. al.* [Schmidt and Wyvill 2005]. In this method the 2D shape sketched by the user is sampled and an implicit approximation is created from the sample points. This is done by fitting a thin-plate spline as a base shape to the sampled points using variational interpolation [Savchenko et al. 1995] [Turk and O'Brien 1999] . One advantage of creating the base shape using variational interpolation is that the resulting implicit field is C^2 continuous, a property needed when the shape is involved in several blending operations [Barthe et al. 2004].

A continuous $2D$ scalar field is created from several distance value samples (p_i, d_i), where p_i describes the position of the sample and d_i its distance to the skeleton formed by the sketched polygon. The thin-plate spline used to create the variational implicit field $f(p)$ is defined in terms of these points weighted by corresponding coefficients w_i combined with a polynomial $P(p) = c_1 p_x + c_2 p_y + c_3$.

$$f(p) = \sum_{i \in N} w_i (\|p - p_i\|)^2 ln(\|p - p_i\|) + P(p)$$

The resulting thin plate spline can then be used as the basis of several different 3D objects:

- sweep along a line
- revolving around an axis
- inflation.

These sketched objects can then be used in the same way as the standard skeletal implicit primitives to create unique 3D shapes. Such unique shapes were not possible to create in previous collaborative environments, especially given this technique's small memory footprint needed to transfer the information.

4 Implementation

Other groupware systems [Greenberg and Roseman 1999] have dealt with the 'large model' problem in several ways. One common approach is screen sharing of single user applications: instead of sending the model, only the screen visuals are transmitted. Key limitations are that users have to take turns (simultaneous input does not really work[Nishino et al. 1999]), and that the model would not be available at all sites for offline use. Another approach transmits only user input, such as mouse movement to keep the model synchronized, since as long as the input across applications remain synchronized, the models constructed at each site would be the same. Such synchronization can be difficult to do in practice, and introduces the 'latecomer' problem, i.e., if a model has already been created ahead of time, either that entire model or the input stream up to that point would have to be transmitted to bring the late entrant up to date. On the contrary, our underlying data structure, the *BlobTree*, has the advantage of a compact representation even for a large model (see the airplane model in Figure 6d), so even sending the whole history does not involve a lot of data transfer.

There remains other cases (detailed below) where transmitting and maintaining a true copy of the model across sites is still a best choice in terms of flexibility and reliability.

4.1 Network Message Layers

To satisfy the aforementioned properties for efficient collaborative systems, our approach maintains a true copy of the model(s) across all the clients. It is for these cases that we advocate our parameterized approach that includes a protocol that can be categorized into several layers, each of them dealing with separate parts of the required communication:

- *system* messages, described in section 4.1.1
- *actions* described, described in section 4.1.2
- *user interface* messages, described in section 4.1.3

We chose a synthesis of techniques from different disciplines as the basis to overcome problems present in several existing distributed modelling environments. In our system, no node is a dedicated server, thus the need for an election algorithm in case the server loses connection is not present. Every node connects to every other node and all messages are sent via multicast to all participating members.

Each message apart from a system message, contains its sending time stamp relative to the start time of the modeling session. These time stamps are mainly needed for synchronization, but they also directly provide one of the additional benefits of our message system, described in section 8.1.

Message types differ in the way they are applied:

- System messages are executed right away when they are read by each host.
- Actions and user interface messages are buffered in between the rendered frames. The buffers are updated at the start of each frame to avoid unnecessary work between frames, and potentially not having the data changed while rendering is in progress. This reduces the computation workload, since the program polygonizes at maximum once every frame.

4.1.1 System Messages

Our system uses Lamport timestamps [Lamport 1978] to provide concurrency between all nodes, as described in section 5. One main objective of the system messages is that all users use the same time base in the messages sent. These time stamps are assumed to be in *coordinated universal time* (UTC) and we assume that every node in the system has consistent time due to a connection to a local time server.

When a new node A connects to one of the nodes B currently in the modeling session, B sends the start time of the session to A. In case node A already has modeled something, the local model is reset and the remote one is to be loaded.

The other main objective of the system messages are handling of all connected users. After A connects to B, B gathers the IP addresses of all its connected nodes and forwards them to A. A starts connections with all the nodes whose data it receives, and confirms to B when this is achieved. Only then A receives the action history (see below) of the current modeling session to create the model and participate in the session.

If one single node loses connection to the system and reconnects, the same procedure applies. The reconnection is handled as if it is a new node connecting, discarding the old information on the reconnected node.

4.1.2 Actions

In our system the term *action* is used for any network message that changes the current shared model. This means that after an action is received and applied the actual model has been changed, compared to a user interface message which is used for immediate feedback and only shows an approximation of the change to be.

Since actions modify the actual model, the following different types of *BlobTree* data objects were defined:

- *primitive objects* with their parameters (e.g. colour),
- *sketched objects* with the parameters (as above) plus the sample points,
- *operator objects*, with optional parameters e.g. the *Ricci Blend Operator* [Ricci 1973] and
- *transformations* (standard affine transformation or warps (bends and taper nodes)).

Both, primitive objects and sketch objects are leaf nodes in the *BlobTree*, whereas transformations have one child node and operators can usually have two child nodes. For primitives, sketched objects and operators, there is only a limited set of potential values (e.g. a sphere or a cube primitive, etc.), thus the main information can be given by setting the exact data objects using its explicit type information. If needed, a limited set of additional unique parameters (such as transformation values) will be transmitted as well. Every action creating a new node in the *BlobTree* gets a unique time stamp, which is part of the message, and actions operating on the existing nodes (operators and transformations), take these IDs as parameters as well. This information can be seen as the minimal representation needed to describe an arbitrary *BlobTree* found in our system: Node type information, IDs and additional parameters.

When a model is created it can be thought of as a series of semantically different tasks:

- **add primitive** and set its parameters,
- **skech object** based on a given control polygon.
- **add operator** combining several nodes of the tree that have parameters depending on the operator,
- **move**, **scale**, **rotate** and **delete** a *BlobTree* node and all its underlying children if present,
- **undo** and **redo** of any action

The actions defined above are independent of their actual implementation in a user interface. For example the delete action can be either triggered by a button click in an application having a CAD like interface or it can be triggered by the user directly using a gesture, in this case crossing out the object in a sketch interface (as in [Schmidt and Wyvill 2005]).

4.1.3 User Interface Messages

A major advantage of our approach is that the *BlobTree* data structures transmitted is small, enabling fine-grained and rapid updates of the scene. This approach also allows us to describe what the users are doing when they are not applying changes to the *BlobTree*. In order to achieve this immediate feedback of *what is going to happen* after the remote user finishes his current task, we incorporated several messages which only describe user input.

We use these messages to update several distinct, non -*BlobTree*-related data:

- camera parameters to have information about everybody's point of view,
- cursor positions to show where remote users are pointing at in 3D space and
- immediate feedback showing the result of a geometric transformation

The *BlobTree* itself is only changed and re-polygonized when the action for the final modification is sent, to avoid unnecessary immediate steps.

Without these messages, changes to the model would just appear at every participant when they are applied to the tree, without any previous feedback. One example of this behaviour would be an object being transported from one location to another. This feedback is needed to communicate changes between all participants in the modelling session. When a user adds a new shape to the scene using sketching, the control points of the sketched shape are transmitted as they are drawn, so the other users are informed at any stage of the drawing process.

The information transmitted in the user interface messages is not used to modify the final tree, but the necessary transformation data is sent separately. This is done, so it is possible to discard all the user interface messages when saving the final model actions, and still have all the necessary data to reconstruct the model.

5 Synchronisation

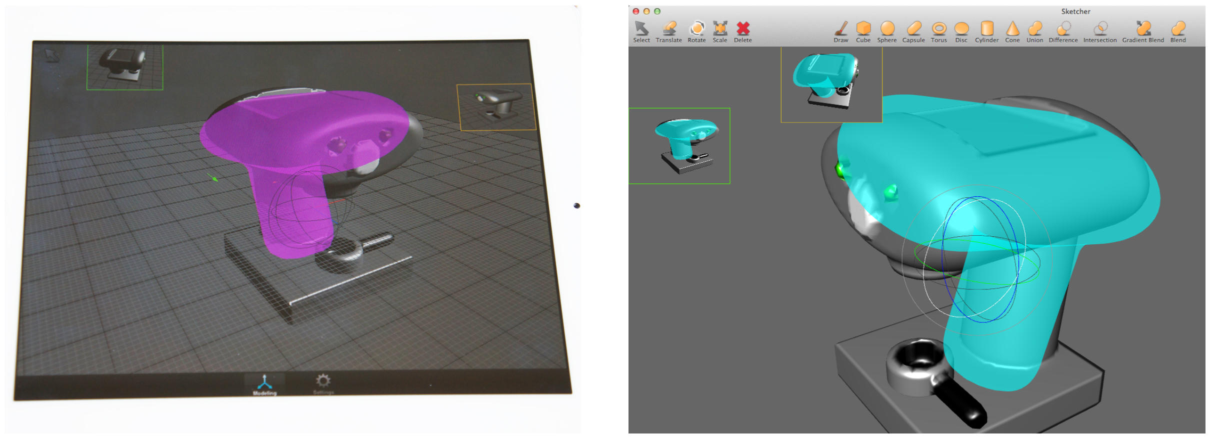
Our synchronization approach is based on *optimistic time stamp ordering* as described by *Kung et. al.* [Kung and Robinson 1981]. The timestamps used are transferred in relative time in microseconds since the session start time, assuming that all the participating users have working clocks that are synchronized via their operating systems. Every action is assigned a time stamp by the host system where it originates and at every participant the actions are applied to the tree in the order of the timestamps.

In case a *latecomer* message arrives and messages with a later timestamp have already been applied, those are rolled back, the latecomer message is applied and all following are redone.

As defined in optimistic time stamp ordering there is a chance of actions conflicting, which in our case can be:

- A node that is already a child node in the tree, cannot be made a child node again, as it would have two parent nodes. If such an action occurs, it will be ignored by our system and the originating user informed about this fact.
- A node that has been deleted cannot be altered. Similar to above such a message will be ignored too.
- In case different users modify the same part of the model our system decides if a potential collision has occurred. A global parameter to the modelling session is, how close in time actions can be applied to the same node by different users. In case actions are too close, they are chosen on a first-come-first served basis, and others are discarded. In our use of the system, a time frame of 1 second has proved to work well, since it can be assumed that an action will be applied at all the other nodes within this time and can be visually registered by all participants.

Every new node in our system is assigned a unique ID using the standard UNIX uuid generator. This generator uses a combination



(a) *iPad* **(b)** *OSX*

Figure 3: *An example modelling session between three users. Both, the iPad (left) and the desktop (right) application show the users, and the modification about to happen (initiated on the desktop).*

of the local mac address and the timestamp to generate a 128bit wide ID, that is considered unique [ISO 1996]. An action creating a new node contains this ID and as a result it is easy to identify the same nodes across the network.

Our system uses a minimum amount of messages as response to actions. There is no need to acknowledge actions, only when they need to be ignored due to a conflict, a message is sent to make sure all other nodes ignore this action too. In these cases, changes are not applied to the model as requested, and users see the model automatically roll back the late change.

6 A Collaborative User Interface

The 3 types of user interface messages (camera parameters, cursor positions, intermediate transformations), are detailed below:

- Camera parameters are used to present the model from the point of view of other users. We use these parameters to render the scene, as seen by the other users into a texture. This texture is then used as the *interactive avatar* for the specific user, displayed as a screen aligned quad at the 3D position of the remote camera. If the remote users camera is outside the current viewing frustum, then the screen aligned quad is clipped to the frustum borders, so it is always present. This *virtual camera* view can be enlarged if necessary by clicking on the avatar. This approach has also proven itself when multiple views of the same model are required.

- The 3D cursor positions of the users are visualized within our scene. If the user is not pointing at any object in the scene, the 3D position is at a constant distance along the view ray of the remote user. An arrow is used to visualize this 3D cursor, with the tip of the arrow being the position transmitted. It is oriented according to the orientation of the corresponding camera.

  In case the remote user is currently sketching a new shape, the transmitted 3D sketch control points are visualized, describing the control polygon of the part of the sketch already drawn.

  When the message to end the sketch action is received, this control polygon is removed from the screen, since it will soon be replaced by the actual sketched object.

- The intermediate transformation results are displayed using the same visuals that are used for transformations, done by the local user. Depending on the type of transformations, certain widgets are displayed at the centre of the current *BlobTree* node. Widgets used locally display an active transformation mode, depending on the chosen motion. Since the desired motion for incoming remote transformations is set by the remote user, only a shadow of the widget is displayed, to illustrate that the current user has no control over the motion. A shadow of the node is also displayed as it is moved, which is used to convey the end position of the object, to every participating member of the modelling session, as soon as the transformation is complete. Otherwise the object would simply be ported from one spot to another, without actually illustrating who did it, and when the transformation was started.

Figure 3 illustrates the above mentioned features, shown for both the desktop and the mobile application. There are two additional users present in the modelling session. On the desktop, both the yellow and the green users look at the scene from their view points, the mouse cursors hidden from the model in the main view. The desktop user interacts with the rotation widget (circles in grey), transforming the rotation of the highlighted object. On the main display on the mobile device the feedback of the translation of the main part of the coffeemaker (highlighted in pink) is transformed via the translation widget (original position in grey, the actual position shown in colour).

7 Access Control

Building a complex model, such as a car, often involves creating several disjoint parts, that might be built by different specialists. In the example of the car, the body would be built by a designer, whereas the engine would be created by an engineer. In some cases it is desirable to have both of the models displayed together, to see if they fit.

For this reason we decided to support several *BlobTrees* in our modelling system. We use the same unique identifiers for identifying the trees in our system as we do for each single node (see section 5). In order to assign each action to the proper tree, these IDs are transmitted with each action. If no tree with the given ID is found in the local modelling session, a new one is created and gets this

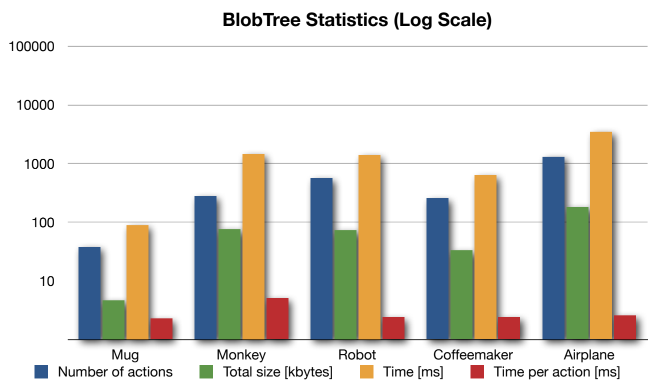


Figure 4: *The network usage characteristics for the BlobTree approach. The y axis has a logarithmic scale*

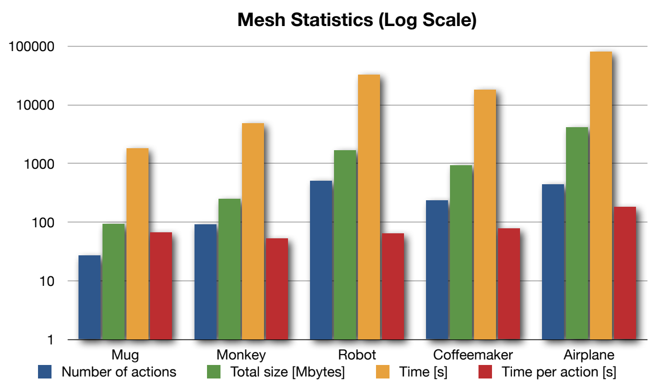


Figure 5: *The network usage characteristics for the mesh approach. The y axis has a logarithmic scale*

ID assigned. Each user chooses his current active tree and he is allowed to switch it at any time in the session. Any action the user takes can only apply to the current selected *BlobTree*, resulting in a lightweight access control system.

Assuming a working communication channel is in place, the designers and engineers can coordinate, which *BlobTree* can be altered by whom. New *BlobTrees* can be added as needed and are displayed as half transparent until selected by the local user. This is done so that they don't obstruct the view of the current active *BlobTree* and to illustrate clearly which objects can be altered.

This lightweight access control system can be extended if needed, by introducing formal access control based on users and user groups, similar to the systems described in section 2. Every node in the *BlobTree* stores ownership information, that can be used to restrict access to the specific node or subtree in the *BlobTree* to either a single user or a group. Whereas the previous work described (e.g. [Han et al. 2003]) uses a central server managing access control, our distributed system would potentially cause problems with a similar mechanism. If, for example, a user/group locks specific parts of the scene to itself, and disconnects, then the locked part will still be locked. Normally an unlock can only be done by the party that did the lock, but if the party is not present anymore, it will continue being locked. Potentially this problem can be solved by introducing timeouts to every lock, but this would mean that locks have to be renewed regularly, resulting in potentially unnecessary communication overhead. Because of this problem we don't use centralized locking and we leave a better locking mechanism as an option to enforce access control to future work.

As mentioned above a lightweight access control mechanism is implemented by splitting the whole scene into several smaller *BlobTrees*, which fortuitously results in an decrease of visualization time, since a change requires repolygonization of only the changed *BlobTree*. If the same scene consisted of a single *BlobTree* with disjoint parts, then a change in one disjoint part would require repolygonizing all other disjoint parts. This specific problem has been solved by *Schmidt et. al.* [Schmidt et al. 2005b], so a combination of both approaches can still result in fast visualisation times. This forms another example of the advantage of maintaining a true copy of the *BlobTree*.

8 Results

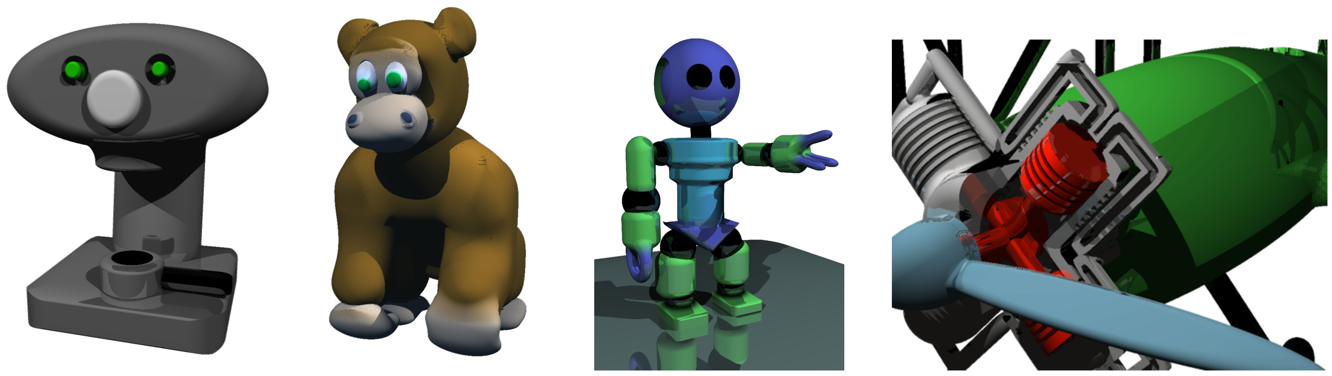
To provide quantitative data of our proposed approach we compared the modelling characteristics against a mesh based synchronization method. The mesh based method sends the mesh at every modelling step to each participant, that requires the model to be changed, thus transmitting the current state of the model, which together with the previous states form the construction history in mesh form. In this evaluation we do not include user interface messages, which are assumed to be equivalent in both approaches. We compared several different modelling sessions of different complexity, shown in Figure 6. For these use cases, we measure the total size of data transferred, time spent transferring this data (latency) and the average time to send a message updating the model. In our test case we simulated an average case 3G network, with 420 kbps uplink and 850 kbps downlink speed.

Figure 4 shows a logarithmic graph for the *BlobTree* case, whereas Figure 5 illustrates the mesh case. Both graphs have logarithmic scale in the y axis, however time for the *BlobTree* case is measured in milliseconds and the data transferred is measured in kilobytes. The mesh graph on the other hand uses seconds and megabytes for the same cases, in order to keep the scale of the graph within a range that can be fit on the page. For the mesh case, there are usually less messages sent, since the mesh will be regenerated at maximum once per frame, thus reflecting the changes of several *BlobTree* actions. Nevertheless, the mesh approach uses a significantly higher amount of data, thus resulting in longer transfer times between the participants. As a result interactivity slows down significantly, given that the average time between meshes is in the 100 second range, whereas the *BlobTree* approach is in the 5 millisecond range for the worst case. The larger the model, the greater the size of the mesh, which increases the average transfer time. In the case of the *BlobTree* the size of a message is independent of the size of the model, as it encodes only the changes in the tree. Sketched objects can have a larger message size due too the variable number of control points (see the monkey model Figure 4, which has many sketch actions and fewer geometric primitives).

8.1 Construction History

There are several advantages of storing the whole construction history over storing only the final model. First of all, by having the construction history of the different parts of the model on hand, the model can be recreated at each step. If a model is highly complex the actions building certain parts can be filtered out, to simplify the model, or in case unnecessary parts were inserted. It is also relatively small and keeping it does not degrade the system.

Since our approach also transmits user input and time stamps, it is possible to playback the whole construction of the model, either in real time, or similar to a video player with changed speed. Several modelling communities teach modelling by using *video* tutori-



(a) *Coffeemaker* (41 *nodes*) **(b)** *Monkey* (64 *nodes*) **(c)** *Robot* (119 *nodes*) **(d)** *Airplane* (810 *nodes*)

Figure 6: *The four models used for comparisons in Figure 4 and 5.*

als, that usually require considerable storage space and bandwidth. Compared to videos our approach needs significantly less storage, even if accompanied by an audio stream, commenting the construction history. If the maker of such a tutorial realizes that he has done something undesired during recording the tutorial, he would need to edit the video using a video editing software. If, in comparison, our approach is used, the undesired messages can be removed using a text editor.

If errors or undesired changes in the final model are found, our approach provides a simple way to determine the user responsible for that part of the model. Since every message can be tracked to its origin, all that needs to be done is to find the appropriate message, resulting in the undesired model, in the history and determining its sender.

9 Conclusion & Future Work

We developed a system based on the *BlobTree* that allows collaborative sketch-based modelling across a network. The network traffic is minimized by using a hierarchical implicit modelling system. We use a sketch based metaphor for direct manipulation of a model, and a layered server-less messaging system that does not require locks for synchronization. This distributed system uses different layers of messages to distinguish between synchronisation and setup (*system messages*), immediate user interface feedback (*Ui messages*) and messages that alter the model(s) under construction (*actions*).

Our application based on the paradigms described in this paper was used to build the four models presented in Figure 6 to illustrate the advantages of our approach: reduced size of transmitted data between all users and optimistic time stamp ordering to avoid a lock-based synchronization approach. Our future work will explore the relationship between model complexity and the use of the message system as described, as well as a detailed comparison with a mesh approach. We did not find a collaborative approach in the literature, that only transfers the change in the mesh, although this idea would reduce the bandwidth for communications, we maintain that a large *BlobTree* can efficiently encode details that would require far more data even in the incremental mesh case.

We have shown, that actions can be recorded for training purposes and also for reviewing the steps that have been done to design a specific part of an object. To control access, we developed a light weight system where disjoint parts of the model can be separated, and every person can only work on one tree, not several at the same time. This reduces the chance of people adding model information to the wrong parts during the session. Apart from the ability to allow for fine grained and rapid updates between all users in the current modelling session our proposed system has several other advantages. We improve on the system most similar to ours, see [Nishino et al. 1999], in that we include a wider variety of primitives and operators. Moreover our approach does not require a centralized server managing the scene and access rights thus not having the problems imposed by this approach. This enables all users to simultaneously access a variety of alternative shape modifications and collaboratively choose the most appropriate result.

Our future work targets a more complex access control mechanism, similar to the one described in [Han et al. 2003], where the amount of details revealed per each participant can be controlled by roles defined in the system.

Acknowledgments

The authors would like to thank NSERC and GRAND for supporting this work.

References

AKKOUCHE, S., AND GALIN, E. 2001. Adaptive Implicit Surface Polygonization Using Marching Triangles. *Computer Graphics Forum 20*, 2, 67–80.

BARBIER, A., AND GALIN, E. 2004. Fast Distance Computation Between a Point and Cylinders, Cones, Line-Swept Spheres and Cone-Spheres. *Journal of Graphics, GPU, and Game Tools 9*, 2, 11–19.

BARTHE, L., DODGSON, N. A., SABIN, M. A., WYVILL, B., AND GAILDRAT, V. 2003. Two-dimensional potential fields for advanced implicit modeling operators. *Computer Graphics Forum 22*, 1, 23–33.

BARTHE, L., WYVILL, B., AND DE GROOT, E. 2004. Controllable binary csg operators for soft objects. *International Journal of Shape Modeling* (Dec.).

BLOOMENTHAL, J. 1994. An implicit surface polygonizer. In *Graphics Gems IV*, P. S. Heckbert, Ed. Academic Press Professional, Inc., San Diego, CA, USA, 324–349.

BLOOMENTHAL, J. 1997. *Introduction to Implicit surfaces*. Morgan Kaufmann.

CHU, C.-H., WU, P.-H., AND HSU, Y.-C. 2009. Multi-agent collaborative 3D design with geometric model at different levels of detail. *Robotics and Computer-Integrated Manufacturing 25*, 2, 334–347.

DE GROOT, E. 2008. *BlobTree Modelling*. PhD thesis, The University of Calgary, University of Calgary.

ELBER, G. 2005. Generalized filleting and blending operations toward functional and decorative applications. *Graphical Models 67*, 3 (Dec.), 189–203.

GREENBERG, S., AND ROSEMAN, M. 1999. Groupware Toolkits for Synchronous Work. In *Computer-Supported Cooperative Work (Trends in Software 7)*, M. Beaudouin-Lafon, Ed. John Wiley & Sons Ltd, 135–168.

HAN, J. H., KIM, T., CERA, C., AND REGLI, W. 2003. Multi-resolution modeling in collaborative design. *Computer and Information SciencesISCIS 2003*, 397–404.

ISO. 1996. Information technology – Open Systems Interconnection – Remote Procedure Call (RPC). *Internatioal Organization of Standardization ISO/IEC 11578.*

KALRA, D., AND BARR, A. 1989. Guaranteed ray intersections with implicit surfaces. *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (July).

KIM, T., CERA, C. D., REGLI, W. C., CHOO, H., AND HAN, J. 2006. Multi-Level modeling and access control for data sharing in collaborative design. *Adv. Eng. Inform. 20*, 1, 47–57.

KUNG, H. T., AND ROBINSON, J. T. 1981. On Optimistic Methods for Concurrency Control. *ACM Transactions on Database Systems 6*, 2 (June), 213–226.

LAMPORT, L. 1978. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM 21* (July), 558–565.

MARION, C., AND JOMIER, J. 2012. Real-time collaborative scientific WebGL visualization with WebSocket. In *Proceedings of the 17th International Conference on 3D Web Technology*, ACM, New York, NY, USA, 47–50.

MISHRA, P., VARSHNEY, A., AND KAUFMAN, A. 1997. CollabCAD: A Toolkit for Integrated Synchronous and Asynchronous Sharing of CAD Applications. In *Proceedings TeamCAD: GVU/NIST Workshop on Collaborative Design, Atlanta, GA, USA*, State University of New York at Stony Brook.

MOUTON, C., SONS, K., AND GRIMSTEAD, I. 2011. Collaborative visualization: current systems and future trends. In *Proceedings of the 16th International Conference on 3D Web Technology*, ACM, New York, NY, USA, 101–110.

NISHINO, H., UTSUMIYA, K., KORIDA, K., SAKAMOTO, A., AND YOSHIDA, K. 1999. A method for sharing interactive deformations in collaborative 3D modeling. In *Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, 116–123.

PASKO, A. A., AND SAVCHENKO, V. V. 1994. Blending Operations for the Functionally Based Constructive Geometry. *CSG 94 Set-Theoretic Solid Modeling: Techniques and Applications, Information Geometers*, 151–161.

PASKO, A. A., ADZHIEV, V., SOURIN, A., AND SAVCHENKO, V. 1995. Function Representation in Geometric Modeling: Concepts, Implementation and Applications. *The Visual Computer 11*, 8 (Oct.), 429–446.

PINELLE, D., GUTWIN, C., AND GREENBERG, S. 2003. Task analysis for groupware usability evaluation: Modeling shared-workspace tasks with the mechanics of collaboration. *ACM Trans. Comput.-Hum. Interact. 10* (Dec.), 281–311.

RAMANI, K., AGRAWAL, A., BABU, M., AND HOFFMANN, C. 2003. CADDAC: Multi-Client Collaborative Shape Design System with Server-based Geometry Kernel. *Journal of Computing and Information Science in Engineering 3*, 2, 170–173.

RICCI, A. 1973. A constructive geometry for computer graphics. *The Computer Journal 16*, 2, 157–160.

SAVCHENKO, V. V., PASKO, A. A., OKUNEV, O. G., AND KUNII, T. L. 1995. Function Representation of Solids Reconstructed from Scattered Surface Points and Contours. *Computer Graphics Forum 14*, 4, 181–188.

SCHMIDT, R., AND WYVILL, B. 2005. Generalized sweep templates for implicit modeling. In *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, ACM, New York, NY, USA, 187–196.

SCHMIDT, R., WYVILL, B., COSTA-SOUSA, M., AND JORGE, J. A. 2005. ShapeShop: Sketch-Based Solid Modeling with the BlobTree. In *Proc. 2nd Eurographics Workshop on Sketch-based Interfaces and Modeling*, Eurographics, Eurographics, 53–62.

SCHMIDT, R., WYVILL, B., AND GALIN, E. 2005. Interactive implicit modeling with hierarchical spatial caching. *SMI '05: Proceedings of the International Conference on Shape Modeling and Applications 2005*, 104–113.

SHAPIRO, V. 1994. Real Functions for Representation of Rigid Solids. *Computer Aided Geometric Design 11*, 2.

SHIRLEY, P., AND MARSCHNER, S. 2009. *Fundamentals of Computer Graphics*. A. K. Peters, Ltd., Natick, MA, USA.

SNYDER, J. 1992. Interval Analysis for Computer Graphics. *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques* (July), 121–130.

SUGIHARA, M., WYVILL, B., AND SCHMIDT, R. 2010. WarpCurves: A tool for explicit manipulation of implicit surfaces. *Computers and Graphics 34*, 3 (June).

TURK, G., AND O'BRIEN, J. F. 1999. Shape transformation using variational implicit functions. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 335–342.

W3C, 2013. Websockets API specification. W3C, Feb.

WYVILL, G., MCPHEETERS, C., AND WYVILL, B. 1986. Data structure for soft objects. *The Visual Computer 2*, 4 (Feb.), 227–234.

WYVILL, B., GUY, A., AND GALIN, E. 1999. Extending the CSG tree. Warping, blending and Boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2, 149–158.