Grasberger, H., Shirazian, P., Wyvill, B. and Greenberg, S. (2011)
Sketch-based collaborative interactive implicit modelling at a distance. Research report 2011-1002-14, Department of Computer Science, University of Calgary, Calgary, Alberta, Canada, May. Includes video figure, duration ~1:19.

Sketch-based Collaborative Interactive Implicit Modelling at a Distance

H. Grasberger [1] and P. Shirazian[1] and B. Wyvill[1] and S. Greenberg[2]

[1]Department of Computer science, University of Victoria, Canada [2]Department of Computer science, University of Calgary, Canada

Abstract

With the increased complexity in state of the art models created using common digital content creation applications, such as AutoCAD, Maya or XSI, the need to have more than one person work on a single model is common. Ideally people would work on the same model at the same time, from several workstations possibly at distant locations. Using a mesh based modelling approach requires synchronizing thousands of triangles over the network between all participating workstations.

In contrast the BlobTree *is based on combining skeletal primitives 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. In this paper we propose a network protocol to allow collaborative, sketch-based implicit modelling using the* BlobTree.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling—Constructive solid geometry (CSG)**

1. Introduction

The origin of our proposed technique lies in the desire to work collaboratively and share highly complex models across the network. Large models are very likely to be constructed by more than one person, particularly for product design where experts in the design of a specific part may be at a different location to designers of adjoining parts.
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 one of the main criteria for our system is its small memory footprint.

Unlike the VRML strategy, where the majority of the 3D geometric data format shared and transmitted on the network is a polygon mesh, our proposed system minimizes network loads by transmitting updates to the hierarchical structure known as the *BlobTree*. Our design sends the information as transparent commands 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 re-built from the commands and visualization of the model is performed locally on each system using available processing resources.

The *BlobTree* by *Wyvill et. al.* [WGG99] is a parametric approach whose modelling paradigms are very similar to Constructive Solid Geometry as first described by *Ricci* in [Ric73]. 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 description of an implicit model.

Both, implicit modes and all the operators that can be used in the *BlobTree* can be described using a small set of parameters, thus being the ideal choice for sending the information across the network. The *BlobTree* can be polygonised for fast visual feedback, or if a high quality image is desired they can easily be rendered using a ray tracing approach. Furthermore the *BlobTree* supports a rapid prototyping approach via sketching as described by *Schmidt et. al.* [SW05] where the basic building block of a sketch primitive can be very small in terms of memory.

The main contribution of this work, is to propose an action based protocol to support co-operative sketch based mod-

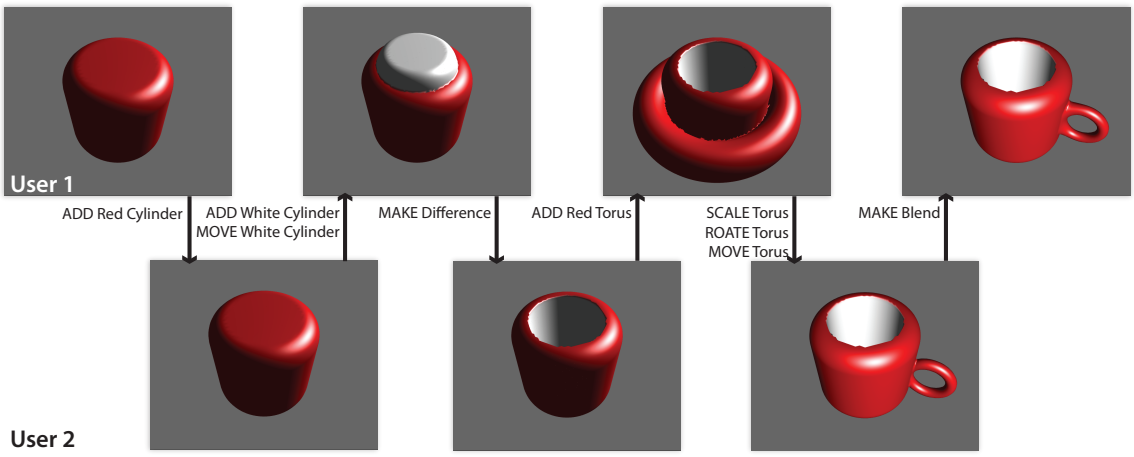


Figure 1: An example modelling session between two users.

elling using the *BlobTree* as shown in figure 1. A second contribution is to introduce a better algorithm for handling concave sketch input in a more robust fashion.

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. This is followed by the description of sketching using variational implicit surfaces including some improvements done in the context of this paper in section 4. We then present the methodology we used to design our protocol in section 5, followed by the advantages over mesh methods in section 6. Finally in section 7 we show some example objects created using our collaborative sketch-based approach together with some performance graphs followed by our conclusions and future work in section 8.

2. Related work

During the past two decades interesting work can be found on concurrent engineering and collaborative design, especially in the field of distributed virtual environments for engineering and manufacturing. Systems such as CollabCAD [MVK97] enables dynamic sharing of the designs across the network amongst multiple designers. Concurrent access to a common design is enabled for viewing and modification. In this system previously designed models are imported into such systems for further manipulation and detailed modifications. The system described in this paper is designed to fill the need for early prototype designs using collaborative sketches.

In a closely related work *Nishino et. al.* [NUK*99] created a collaborative modelling environment to enable design of implicit objects. 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 deformation on the model each participant should request 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 tree data to the session server. The major problem with this system is that only one designer can access the model for modifications at any given time.

Han et. al. [HKCR03] proposed a collaborative design framework which enables CAD engineers to work on models concurrently. The entire assembly design is partitioned into security features which can be grouped together. An access matrix defines what activities can be accomplished on each security feature or group and to what extent that entity is visible to different engineering roles. Per each security feature, a multi-resolution mesh hierarchy is created.This way the rate of details exposed to different participants can be controlled. 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.

Distributed sketching has been a topic of long interest in the Computer Supported Cooperative Work (CSCW) and groupware community. While most reported systems are of simple 2d sketches, the human and social factors underlying distributed interaction apply equally to 3d drawings. 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 [PGG03]. In brief:

- *Explicit communication* occurs not only through spoken and written messages, but by gestural messages, deictic references 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 a cubes in 3d space and camera items, implemented as arrows to show their location and orientation (see the accompanying video).

3. The *BlobTree*

The *BlobTree* extended existing skeletal implicit surface modelling techniques [Blo97] by introducing a unified structure in which nodes could represent arbitrary blends between objects as well as Boolean operations and warping at a local and global level. Geometric transformation matrices are also stored in the tree so the data structure also represents a scene graph. When the tree is traversed, the operations, including visualization only depend on a field-value and a gradient returned for an arbitrary point in space.

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 [WMW86]. Bloomenthal published a popular implementation of the uniform grid method in [Blo94], which in addition, overcame ambiguities using tetrahedral decomposition. A more efficient algorithm was published in [AG01].

The sketch-based system of [SWCSJ05], for efficiency trades accuracy for speed by storing cache nodes in the *BlobTree* [SWG05]. For accurate visualization but in general non-interactive applications, ray tracing can be employed using interval analysis [Sny92], or Lipschitz approaches such as [KB89].

3.1. Skeletal Primitives

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 [dG08] or ShapeShop [SWCSJ05]. They are ideally suited to prototype shapes of arbitrary topology [Blo97]. 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. Usually the skeleton itself is a very simple shape such as a point or a line, but also more complex skeletons can be used. To create a skeletal primitive the distance-field has to be computed as described in [BG04]. This is done by computing the distance to the skeleton for each point in the volume encapsulating the final shape. The distance function is defined as $d_S : \mathbb{R}^3 \to \mathbb{R}$. As a result the distance field is a volume of scalar values which is not bounded as the distance itself can be infinitely large.

In the next step the distance field d_S has to be modified by a field function bound to a finite range. This field function is defined as $g : \mathbb{R} \to \mathbb{R}$ and as a result the skeletal primitive is formed by applying this function to the given distance d_s. Usually the function maps the distances to the range $[0,1]$. The implicit function of one skeletal primitive is $f_(p) = g(d_S(p))$. The most widely used field function was developed as a simplification of the original Soft Objects [WMW86] field function called the Wyvill field function. It maps a distance d to the field value $g(d)$ by the following formula $g(d) = \left(1 - \frac{d^2}{r^2}\right)^3$. In this formula r is a constant value that states the distance where the field value equals zero. The main advantage of this field function is that it is C^2 continuous. A discussion of field function appears in [SM09].

After applying the field function to the distance field we call the resulting field the potential field. The surface is defined as the locus of points that have a field-value equal to some chosen iso-value. By defining an *iso*-value c it is possible to construct the surface of the shape and classify the surrounding space into those points inside the surface $(f(p) > c)$ and outside $(f(p) < c)$. The chosen iso-value depends on the exact form of $G(d)$, for example $c = 0.5$ see [Blo97].

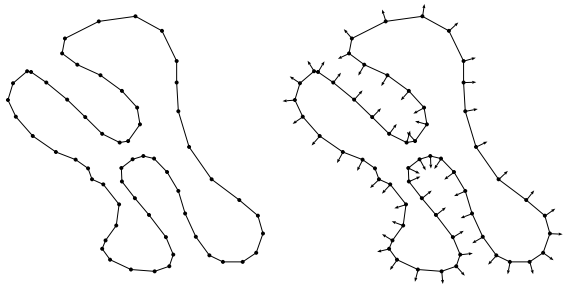
3.2. Blend Operators

When an operator is applied to skeletal primitives, it is actually performed on the field-values f. 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* [Blo97]: $f_R(p) = \sum_{n \in N} f_n(p)$ 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 [BDS*03, BWdG04] or the blending functions that are based on R-functions [Sha94, PASS95, PS98], 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 [Elb05].

4. Sketching

The implementation of sketching follows the approach described by *Schmidt et. al.* [SW05]. In this method the shape sketched by the user is sampled at a lower resolution and an implicit approximation is created from the sample points. This is done by fitting a thin-plate spline to the sampled points (see figure 2a) using variational interpolation [TO99]. 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. If the sample points are on the control polygon the distance is the set iso-distance (the distance where our field function $g(d) = iso\_value$). 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 $P(p) = c_1 p_x + c_2 p_y + c_3$.

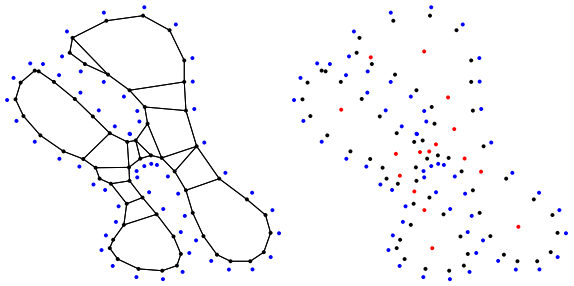


(a) Control Polygon and Sample Points (b) Normals at each Vertex

Figure 2: The polygon formed by the sample points and the normals at the actual length for displacement.

$$f(p) = \sum_{i \in N} w_i (\|p - p_i\|)^2 ln(\|p - p_i\|) + P(p) \qquad (1)$$

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 [BWdG04]. In order to create the thin-plate spline, additional points have to be computed, both inside and outside the basic shape, in order to bound the implicit field. According to *Schmidt et. al* these additional sample points are taken by displacing the control polygon along the vertex normals, in both outside (figure 2b) and inside directions with their distance values d_i being the length of the displacement along the normal. This can lead to problems in situations involving concave polygons, since the displaced polygon could intersect with a polygon notch, thus producing sample points with wrong polygon distances. In addition it can happen that the inside displaced points actually lie outside the original polygon as well.



(a) Control Polygon and Convex Decomposition (b) All Sample Points

Figure 3: The convex decomposition and the final sample points used to build the thin-plate spline.

For the above reasons we take a different approach to create the boundary points. In order to support base shapes of very high concave detail (e.g. sketching the tentacles of an octopus, or branches of a tree), it is necessary to compute the maximum distance, the sample points can be displaced to the outside, without producing self intersections. In order to achieve this, we compute an estimate of the minimum distance d_{min} between all pairs of vertices of the original polygon, to find an approximate distance between notches. We can assume that neighbouring vertices cannot be part of two notches at once, but could be closer than two notches (see the tip of the top left notch in figure 2a), so we discard the direct neighbourhood of each vertex in the distance computation. Although this algorithm is $O(n^2)$, the number of points is relatively small so there is no loss in interactive performance.

The outside offset curve is now produced by displacing each control point along its normal (figure 2b):

$$p_{i_o} = p_i + n_i * \frac{d_{min}}{2} \qquad (2)$$

The displacement is half of d_{min} ensuring that the offset

curve is not self intersecting and as a result does not intersect the original control polygon as well. One criteria for *BlobTree* primitives is that they are bounded, meaning the implicit field reaches 0 at a given distance (in most cases $d = 1$) from the skeleton. To satisfy this criteria, we place additional control points on the border of our sketch area, that have distances $d > 1$.

A different technique is used to generate the sample points inside the sketch perimeter. If the control polygon is convex, then the centre point of it is chosen as the only sample point, with its distance being $d = 0$, meaning it lies directly on the skeleton. It is not guaranteed that the centre point of a concave polygon lies within the polygon boundaries, thus it is not a candidate for a sample point. Furthermore, if only an offset curve is used for sampling the interior of the polygon, it is not guaranteed that the resulting field reaches field values $f = 1$. For these reasons we decided to spend additional computation on concave polygons.

The medial axis [Blu67] of a polygon describes a set of points which can be interpreted as the skeleton of our control polygon. A sample point on the skeleton has the corresponding distance $d = 0$. Since it is not trivial to find points along the medial axis of a concave polygon we first of all compute its *Approximate Convex Decomposition* as described by *Lien et. al.* [LA06] which can be seen in figure 3a. After computing the convex sub-polygons of the desired shape, we calculate their centroids, which we use as an approximation to the medial axis of the shape.

Figure 4: The implicit field created using a thin plate spline multiplied by a cosine function.

The skeletal primitives in our system yield a field value of unity on the skeleton and drop to zero at the limit of their influence on the field. For our case it proved to be beneficial to compute the minimum distance from each of the centroids to the control polygon, and assign the distance $d = 0$ to the centroid with the maximum distance in order to normalize the distance in the range [0:1]. The other centroid distances are weighted according to $d_{i_{corr}} = \frac{d_i}{d_{max}} * 0.5$. This ensures that there is at least one sample point that has the distance value 0 resulting in the fieldvalue 1. Figure 3b shows all sample points: on the control polygon in black, outside in blue and inside in red and figure 4 shows the resulting 2D field.

5. Methodology

Other groupware systems have dealt with the 'large model' problem in several ways (e.g. [GR99]). 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), and that the model would not be available at all sites for offline use. Another approach uses replicated applications where only users' input actions are transmitted between them: as long as the input actions across applications remain synchronized, the models constructed at each site would be the same. However, such synchronization can be difficult to do in practice. It also 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. While other methods exist, there remains many cases where transmitting and maintaining a true copy of the model across sites is still a best choice in terms of flexibility and reliability. It is for these cases that we advocate our paramaterized approach.

5.1. Structure of the BlobTree

The first step to designing the underlying protocol used for our system was to identify the different classes of data that are stored in the *BlobTree*:

- *primitive objects* with their object colours,
- *sketched objects* with their object colour and the sample points,
- *operator objects*, with optional parameters e.g. the *Ricci Blend Operator* [Ric73] and
- *transformations* (standard affine transformation, warps, bends and taper nodes).

Both, primitive objects and sketch objects are leaf nodes in the *BlobTree*, whereas transitions have one child node and operators can usually have two or more child nodes. Adding child information to the types of objects creates the overall tree structure. This information can be seen as the minimal representation needed to describe an arbitrary *BlobTree* found in our system.

5.2. Interactions on the BlobTree

In the next step of designing the protocol we identified the possible interactions involved with modifying the *BlobTree*. This resulted in several *actions*:

- *add primitive* and set its colour,
- *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,
- *change parameters* of a node (e.g.. colour, Ricci Blend parameters, add/remove/change a control point of a sketched object) and
- *delete* a node and its subtree if present.

In order to provide functionality to assign ownership of certain parts of the tree to users we added the actions *lock* and *unlock* which are triggered by selection and deselection through the user interface (see section 5.3).

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 crossing out the object in a sketch interface (as in [SW05]). Selection on the other hand could be done either via a click, or circling the desired object.

As a result actions create an additional layer between the data representation and the user interface. Only this layer needs to be unified to enable programs, that differ in both, user interface and internal data representation, to communicate.

In addition to the actions effectively changing the *BlobTree* we incorporated a set of actions only describing user input for immediate feedback. This type of action is used to update camera and cursor positions, and transmit the intermediate transformation results, a user sees on his local machine across the network. The *BlobTree* is only changed and repolygonized when the non-UI action for the final modification is sent.

5.3. Synchronisation

Actions sent across the network have to be *acknowledged* by each receiving participant if the execution of the action was successful. If an action cannot be acknowledged by one of the participants, a pre defined error code is sent to all remaining users, which then have to undo the action. This ensures, that all participants work with the same state of the *BlobTree*.

Every primitive, sketch or operator node has a unique id assigned at the time of creation which is sent in the action message. To avoid the situation, where the created id does not match the received, the receiving participant of an action creating a new *BlobTree* node only acknowledges this action, if the ids match.

In order to avoid multiple participants changing the same nodes simultaneously, nodes are locked when selected on one host. If the selected node and all child-nodes are not locked, the lock message gets sent. The lock message fails with an error code, if this node, or a child-node is already locked at a different participant. This triggers unrolling of the lock action at each participant that was able to execute it. If the lock was successful then the participant sending the lock is allowed to change it until an unlock message is sent.

6. Advantages

6.1. Construction History

Actions describe all the interaction on the *BlobTree*. This includes the current structure and also how the *BlobTree* evolved over time. The construction history supports undo actions, either as a direct result of an undo command but also for automatic rollback to restore the state of the *BlobTree* in the case of some action failing.

Furthermore action chains can be used to show how a model was created. In several communities it is very common to record a so called *video tutorial*, which usually requires considerable storage space and bandwidth. Compared to videos our actions need less storage, and could be accompanied by an audio stream, thus allowing for teaching modelling to novice users.

Lastly, if an error is to be found in a model, having the history enables designers to find out, when that error occurred and potentially who is responsible for it as well.

6.2. Size compared to Mesh Approaches

In our approach we do not need to store the mesh of a highly complex model, but sufficient information to construct the *BlobTree*. This is the description of the primitives and how these are combined to the final model. Figure 5 shows a comparison of the number of nodes, the size of the whole action history, the size of a polygonised mesh and the number of vertices found in this mesh. It should be noted that the *y* axis of the graph is a logarithmic scale.

Furthermore, changing the *BlobTree* is easier to describe in terms of command size than changing a mesh, e.g. removing one primitive from the *BlobTree* compared to removing 300 vertices from the mesh.

Previously we mentioned the construction history, which can take up a little more space than the actual structural description of the final model. In theory it would be possible to have a similar history for mesh models, but for the same reason that the space requirements for mesh models are larger, a construction history on mesh models would increase the storage requirements by the same relative amount.

These features could become very interesting in the future with the advent of web based 3D games using WebGL. Depending on the game there will be a large need to transport content across the net, which could potentially save a lot of bandwidth if our technology was used. Similar to meshes

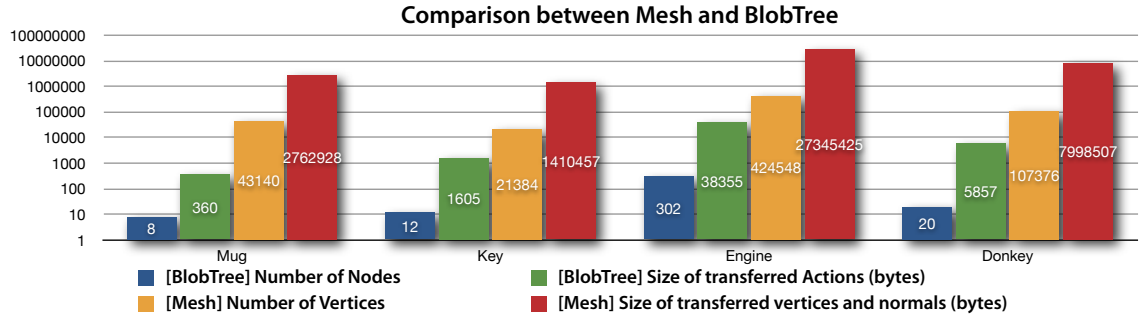


Figure 5: Comparison between the number of nodes, the number of vertices, the size of the actions and the size of the mesh of several models, shown below.

the actions described in our system can be compressed. large savings can be achieved when the whole construction history is sent across the network in a compressed format, which for example for the our engine model decreases the size by about a factor of 6 (38,355 vs 6,497 bytes) using standard zip compression.

6.3. Action Concept is Format Independent

For several reasons our action format is not encapsulated in a container format, but it mainly follows the syntax of a simple human command: `do what how`, e.g. `ADD SPHERE 0 VEC3(1.0, 1.0, 1.0)`. Since a container format only describes how messages are stored or sent it is very easy to build a version of our protocol, that could be integrated into several model format standards such as Collada or X3D.

A possible way to describe the same action as above using XML can be seen in listing 1.

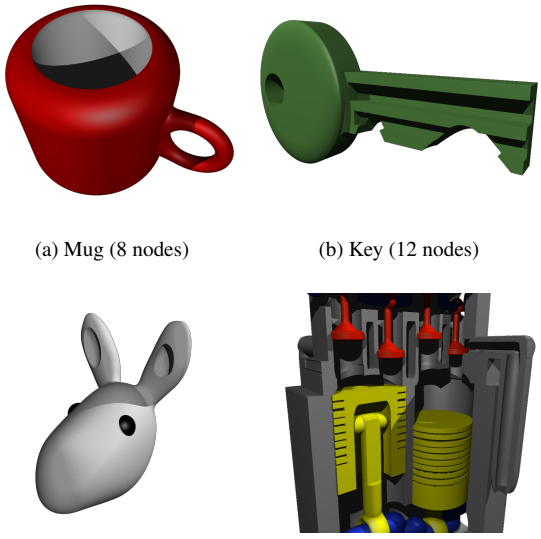
Listing 1: An example of how an action could be realized in XML

```
<add type="sphere" id=0>
  <colour r=1.0, g=1.0, b=1.0 />
</add>
```

7. Results

Figure 5 compares four models (see figure 6) created in our system to their corresponding mesh representation. The mesh representations were extracted from finished the models and stored as an obj file. Actions are saved as human readable text similar to an obj file storing mesh components in formatted text.

The green columns show the sizes of the actions sent across the network for each model; these values are comparable to the sizes of the polygon meshes transferred (red columns). It has to be pointed out that our mesh data measurement only reflects the final product, not all of the intermediate steps involved. On the contrary the blue columns show the complexity of the model measured in terms of number of tree nodes, which can be compared to the number of vertices (orange). These graphs show, that our compact action mechanism results in less bytes transferred, to describe a model in greater detail.



(a) Mug (8 nodes) (b) Key (12 nodes)

(c) Donkey (20 nodes) (d) Engine (302 nodes)

Figure 6: Our 4 models used for comparisons in figure 5.

8. Conclusion & Future Work

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. The sketching mechanism incorporated in our system using variational implicit objects enables designers to create a wider range of shapes as compared to Nishino's collaborative Modelling technique [NUK*99].

Our future work targets a more complex access control mechanism, similar to the one described in [HKCR03], where the amount of details revealed per each participant can be controlled by roles defined in the system. One issue

with synchronization not dealt with in this paper is the case when a participant loses the connection and reconnects after a couple of actions are sent. In addition we plan to include less pessimistic concurrency control to avoid possible delays due to our locking mechanism.

We are planning to introduce sketched objects that allow for holes in the sketches for more flexibility when doing rapid prototyping. Furthermore we want to extend our list of actions to incorporate warp, taper and bend operators to have a wider variety of possible object deformations.

Acknowledgments

The authors would like to thank NSERC and GRAND for supporting this work and Jyh-Ming Lien for providing the source code for the ACD algorithm.

References

[AG01] AKKOUCHE S., GALIN E.: Adaptive implicit surface polygonization using marching triangles. *Computer Graphics Forum 20*, 2 (2001), 67–80. 3

[BDS*03] BARTHE L., DODGSON N. A., SABIN M. A., WYVILL B., GAILDRAT V.: Two-dimensional potential fields for advanced implicit modeling operators. *Computer Graphics Forum 22*, 1 (2003), 23–34. 4

[BG04] BARBIER A., GALIN E.: Fast distance computation between a point and cylinders, cones, line-swept spheres and cone-spheres. *Journal of Graphics, GPU, and Game Tools 9*, 2 (2004), 11–19. 3

[Blo94] BLOOMENTHAL J.: *An implicit surface polygonizer*. Academic Press Professional, Inc., San Diego, CA, USA, 1994, pp. 324–349. 3

[Blo97] BLOOMENTHAL J.: *Introduction to Implicit Surfaces*. Morgan Kaufmann, ISBN 1-55860-233-X, 1997. Edited by Jules Bloomenthal With Chandrajit Bajaj, Jim Blinn, Marie-Paule Cani-Gascuel, Alyn Rockwood, Brian Wyvill, and Geoff Wyvill. 3, 4

[Blu67] BLUM H.: A Transformation for Extracting New Descriptors of Shape. In *Models for the Perception of Speech and Visual Form*, Wathen-Dunn W., (Ed.). MIT Press, Cambridge, 1967, pp. 362–380. 5

[BWdG04] BARTHE L., WYVILL B., DE GROOT E.: Controllable binary CSG operators for soft objects. *International Journal of Shape Modeling* (Dec 2004). 4

[dG08] DE GROOT E.: *BlobTree Modelling*. PhD thesis, The University of Calgary, 2008. 3

[Elb05] ELBER G.: Generalized filleting and blending operations toward functional and decorative applications. *Graphical Models 67*, 3 (Dec 2005), 189–203. 4

[GR99] GREENBERG S., ROSEMAN M.: *Groupware Toolkits for Synchronous Work*, vol. ISBN 0471 96736 X. John Wiley & Sons Ltd, 1999, ch. 6, pp. 135–168. 5

[HKCR03] HAN J., KIM T., CERA C., REGLI W.: Multi-resolution modeling in collaborative design. *Computer and Information Sciences-ISCIS 2003* (2003), 397–404. 2, 7

[KB89] KALRA D., BARR A.: Guaranteed Ray Intersections with Implicit Functions. *Computer Graphics (Proc. SIGGRAPH 89) 23*, 3 (July 1989), 297–306. 3

[LA06] LIEN J.-M., AMATO N. M.: Approximate convex decomposition of polygons. *Computational Geometry 35*, 1-2 (2006), 100 – 123. Special Issue on the 20th ACM Symposium on Computational Geometry. 5

[MVK97] MISHRA P., VARSHNEY A., KAUFMAN A.: CollabCAD: A Toolkit for Integrated Synchronous and Asynchronous Sharing of CAD Applications. In *Proceedings TeamCAD: GVU/NIST Workshop on Collaborative Design, Atlanta, GA, USA* (1997), pp. 131–137. 2

[NUK*99] NISHINO H., UTSUMIYA K., KORIDA K., SAKAMOTO A., YOSHIDA K.: A method for sharing interactive deformations in collaborative 3D modeling. *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '99* (1999), 116–123. 2, 7

[PASS95] PASKO A., ADZHIEV V., SOURIN A., SAVCHENKO V.: Function Representation in Geometric Modeling: Concepts, Implementation and Applications. *The Visual Computer 11*, 8 (Oct 1995), 429–446. 4

[PGG03] PINELLE D., GUTWIN C., GREENBERG S.: Task analysis for groupware usability evaluation: Modeling shared-workspace tasks with the mechanics of collaboration. *ACM Transactions on Human Computer Interaction - ACM TOCHI 10*, 4 (December 2003), 281–311. 2

[PS98] PASKO A. A., SAVCHENKO V. V.: Blending Operations for the Functionally Based Constructive Geometry. *CSG 94 Set-Theoretic Solid Modeling: Techniques and Applications, Information Geometers* (Dec 1998), 151–161. 4

[Ric73] RICCI A.: A constructive geometry for computer graphics. *The Computer Journal 16*, 2 (1973), 157–160. 1, 5

[Sha94] SHAPIRO V.: Real Functions for Representation of Rigid Solids. *Computer-Aided Geometric Design 11*, 2 (1994). 4

[SM09] SHIRLEY P., MARSCHNER S.: *Fundamentals of Computer Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2009. 3

[Sny92] SNYDER J.: Interval analysis for computer graphics. *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques* (Jul 1992). 3

[SW05] SCHMIDT R., WYVILL B.: 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* (New York, NY, USA, 2005), GRAPHITE '05, ACM, pp. 187–196. 1, 4, 6

[SWCSJ05] SCHMIDT R., WYVILL B., COSTA-SOUSA M., JORGE J. A.: Shapeshop: Sketch-based solid modeling with the blobtree. In *Proc. 2nd Eurographics Workshop on Sketch-based Interfaces and Modeling* (2005), Eurographics, Eurographics, pp. 53–62. Dublin, Ireland, August 2005. 3

[SWG05] SCHMIDT R., WYVILL B., GALIN E.: Interactive implicit modeling with hierarchical spatial caching. In *SMI '05: Proceedings of the International Conference on Shape Modeling and Applications 2005* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 104–113. 3

[TO99] TURK G., O'BRIEN J. F.: Shape transformation using variational implicit functions. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., pp. 335–342. 4

[WGG99] WYVILL B., GUY A., GALIN E.: Extending the CSG tree. warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2 (Jan 1999), 149–158. 1

[WMW86] WYVILL G., MCPHEETERS C., WYVILL B.: Data Structure for Soft Objects. *The Visual Computer 2*, 4 (February 1986), 227–234. 3