www.cadanda.com
Computer-Aided Design AND APPLICATIONS

Enhancements for Improved Topological Entity Identification Performance in Multi-user CAD

Ammon Hepworth[1], Daniel Staves[2], Logan Hill[3], Kevin Tew[4], C. Greg Jensen[5] and W. Edward Red[6]

[1]Brigham Young University, ammon.hepworth@byu.edu
[2]Brigham Young University, danstaves@gmail.com
[3]Brigham Young University, alhill29@hotmail.com
[4]Brigham Young University, kevin_tew@byu.edu
[5]Brigham Young University, cjensen@byu.edu
[6]Brigham Young University, ered@byu.edu

ABSTRACT

Multi-user CAD allows designers to simultaneously work on a model, allowing designs to be realized at a much faster rate than ever before. In a replicated, simultaneous multi-user CAD system, it is critical that models be consistent between clients. A major component of model consistency is ensuring references to topological objects be the same on all clients in the same part. Previous methods are inefficient for models with a large number of faces and edges. This paper presents enhancements over previous methods which more efficiently identify faces and edges through lazy naming as well as caching and normalizing topological entity data. The implementation and results of these enhancements show significant time savings compared to an eager naming method.

Keywords: persistent naming, multi-user CAD, collaborative engineering.

1. INTRODUCTION

Simultaneous multi-user CAD allows a geographically dispersed team to work concurrently on the same model as opposed to serially. A replicated multi-user CAD system employs a client-server (CS) architecture where each client has a replicated instance of the CAD part. As users model, operational data is extracted from clients and sent through a centralized server to remote clients. Client models are updated as remote operations are received from the server. This enables simultaneous modeling and real-time updates from several remote clients.

One central issue in a replicated, multi-user CAD system is ensuring that references to topological entities are consistent between clients. Feature-based CAD systems create features that parametrically reference topological entities which include faces, edges and vertices. References to the topological entities in a CAD part on one client must be the same on all clients to ensure dependent features are applied to the same topological entity on all clients. For example, a fillet feature operation applied to an edge in one client needs to be applied to the same edge in all other replicated client models. If the system fails to keep names of features and entities persistent between clients, models will become inconsistent, causing errors to occur. This issue is referred to in the literature as the persistent naming problem [12].

In a previous paper we discuss a persistent naming method for a multi-user CAD system in which topological entities of the geometry kernel are not returned in a predictable order [9]. This is the case with at least one major geometry kernel (namely Siemens Parasolid). Since faces and edges cannot be identified by the order in which they are returned, the method uniquely identifies them by their geometric properties. These properties are mapped to a unique identifier and stored in the operational data forwarded to other clients. When the clients receive the remote operation, they identify the bodies, edges, and faces by matching the geometric properties to their corresponding identifier. This method is referred to as eager naming because it identifies all the topological identities as soon as they are created.

Computer-Aided Design & Applications, 12(5), 2015, 627–638, http://dx.doi.org/10.1080/16864360.2015.1014742

Taylor & Francis
Taylor & Francis Group

1.1. Feature Creation Performance Problem

While the eager naming method does ensure consistent identification for all clients, it can take a considerable time to identify the topological entities of a feature creation operation which generates many bodies, edges, or faces. Fig. 1 shows an extrusion of a grate that creates 406 faces and 1212 edges. The extrude feature creation operation for the grate takes approximately 5 min. to identify all the topological entities using the eager method and is consider excessive. Each client will need to perform this same naming algorithm on their local machine, causing interruption to the user workflow by forcing the user to wait until the naming operation is complete so local client work can continue.

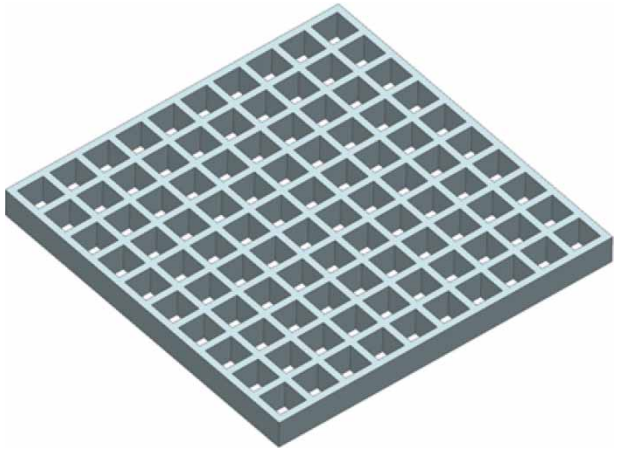


Fig. 1: The extrusion of the grate takes about 5 min. to identify faces and edges.

1.2. Feature Edit Performance Problem

The eager naming method further interrupts the modeling process when a feature edit operation is performed. A feature edit operation causes not only the selected feature to change, but also any dependent features to update as well. Each updated feature can cause any number of faces and edges to change shape, size, or location. Because the eager naming method stores topological entity data together with feature operation data, the geometric properties of every topological entity of every dependent feature must be resent to the server with the edit operation data. This results in a large amount of data being sent between clients, as well as a significant amount of time being spent re-identifying the topological entities of the dependent features.

The examples shown in Fig. 2 illustrate the feature edit performance problem well. Image A shows a gear

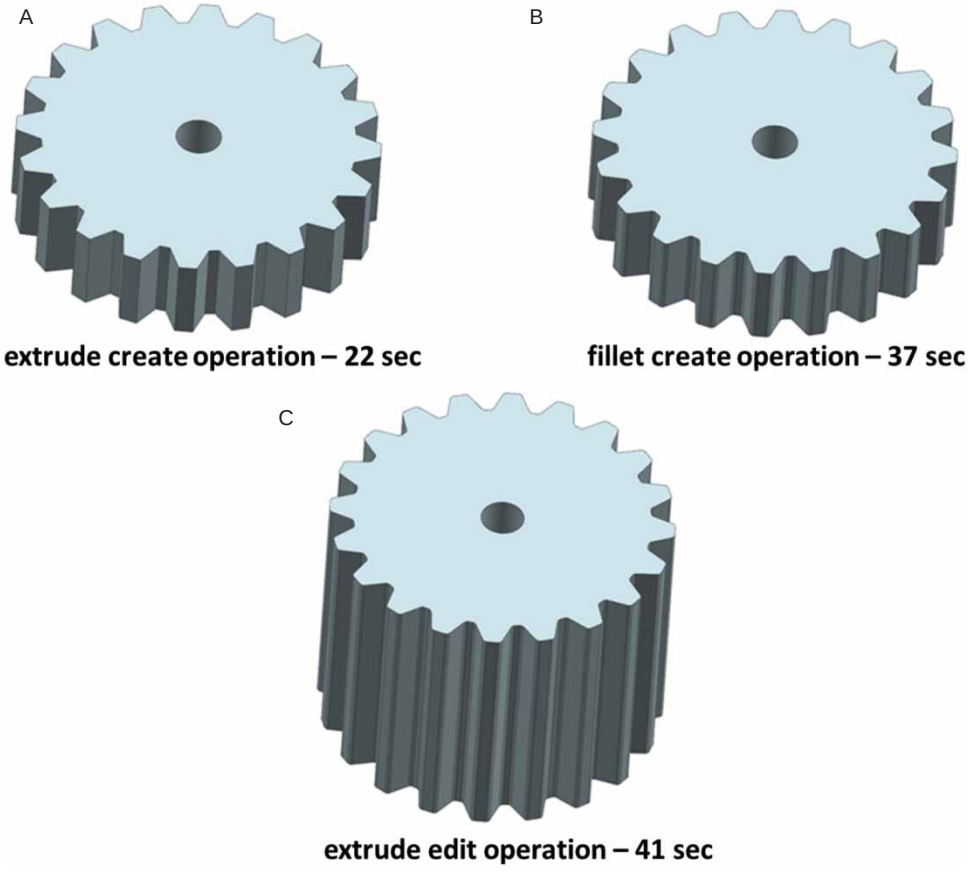


Fig. 2: A: Gear extrusion; B: Fillets added to gear sides, C: Thickened extrusion.

which has 20 teeth and a hole through the center. The gear with 83 faces and 242 edges is created by an extrusion operation which takes about 22 sec. to perform with the eager naming logic. Image B shows fillets applied to all 80 of the vertical edges on the side of the gear creating 80 additional faces and 240 additional edges taking about 37 sec. Image C shows the extrusion edited to thicken the gear, changing the size of all the edges and faces in the model. The extrusion edit takes about 41 seconds to perform, which is longer than the time it takes to perform the fillets. The time to perform this operation includes the time for extracting the geometric data for each face and edge in both the extrude and the dependent fillet operations.

1.3. Performance Enhancements

The eager method, though effective, causes unnecessary delay for the most basic of models. For large models or assemblies, the eager naming method severely hinders the multi-user, collaborative CAD experience. We present an approach based on the geometric identification principles found in the eager naming method, but overcomes its shortcomings with three major enhancements.

The first enhancement solves the feature creation performance problem using lazy naming. Lazy naming shifts the identification paradigm from naming each topological entity immediately after its creation, to a scheme which names topological entities just before they are referenced by dependent features. The method is termed lazy because it waits to identify topological entities when they are actually needed. Lazy naming dramatically improves feature creation time because identification does not occur until a topological entity is actually referenced.

The second enhancement solves the feature edit performance problem. It persistently stores the topological entity data separate from the feature data in the database and stores a cache of the same data in a client entity dictionary on each client. This second enhancement eliminates the requirement of the eager method to extract the geometric properties of all dependent features after a feature edit.

The third enhancement additionally helps improve performance for edit operations in the Siemens NX multi-user prototype. This implementation enhancement is called enhanced geometric property extraction and uses a more efficient method to directly query the geometric properties of faces and edges. This is accomplished by taking advantage of the internal NX identification system (Journal ID's) which more quickly extracts the geometric properties of the entities. These three enhancements dramatically decrease the time for naming in feature creation and editing for multi-user CAD, thus improving the overall multi-user CAD experience.

2. BACKGROUND

2.1. Existing Collaborative CAD Implementations

Research in collaborative CAD has been ongoing since the mid-90s with two main architectures emerging: centralized and replicated. A centralized architecture utilizes a central server which executes modeling operations received from remote clients. After performing operations, the server passes the resultant geometric data to clients for visualization and interaction. In this approach, the server performs the computation for all geometric operations and clients are used for visualization and user interaction. WebSPIFF [4], NetFeature [24], CADDAC [25], CollFeature [33] and WebCOSMOS [37,38] are all examples of a central architecture. The advantage to this approach is that there is only one copy of the model so there is no need to keep models consistent between clients. Persistent naming for these types of collaborative CAD systems are inherently different than that of replicated systems since there is only one copy of the model. A downside to this architectural approach is that a large amount of data needs to be sent over the network [12]. Another disadvantage is that geometric algorithms must be multi-threaded to enable timely processing of operations from multiple users. To date, the authors are not aware of any implementation of this approach based on commercial CAD software.

Conversely, replicated collaborative CAD systems have copies of the model data on each client which are required to stay in sync. Operation data is sent between clients via a network architecture. ARCADE [31], CSCW-FeatureM [32], CollIDE [23] and RCCS [12] are examples of replicated CAD systems built at the kernel level. Examples of systems that interface with commercial single user systems are TOBACO [8], CallabCAD [21], COCAD [16]. In addition, multi-user systems developed at the NSF Center for e-Design, BYU site are built as direct plug-ins to major commercial single user systems including Siemens NX, Dassault Systemes CATIA and Autodesk Inventor. These plugins are respectively named NXConnect, CATIAConnect and InventorConnect [26–28]. The replicated approach is easier to implement with existing commercial CAD systems because it does not require multi-threaded algorithms for parallel operations to be performed and most commercial CAD system APIs are single threaded [28]. Another advantage to this architecture is that it does not require large data to be sent over the network because operation data is more concise than visualization data. However, the main challenge with replicated systems is data consistency and conflict management between clients due to the lack of a centralized model.

2.2. Persistent Naming

Persistent naming has been a problem for researchers since the beginning of history-based parametric solid

modeling. This is because two different models represent the part. One is the parametric model, which consists of modeling operations, and the other is the geometric model [12]. The main issue has to do with keeping the faces, edges and vertices of the geometry model consistent with the parametric model when it is reevaluated from the operational history. Directly using computer memory pointers for identification is not a valid technique because they are transient. Simple enumeration methods do not always work because model edits change topology and the enumeration is no longer legitimate [17]. Several authors have presented solutions to the persistent naming problem in single-user CAD [1–3, 5–7, 20, 34,35].

In replicated, collaborative CAD systems, persistent naming is concerned primarily with uniquely identifying topological entities on various remote clients. Jing et al. recognize that naming topological entities directly from modeling history may not be a valid solution if operations on various clients may be performed in a different order. They present methods to solve this by naming topological entities of a given feature in a consistent order [13, 18,19]. One assumption that was made in their implementation is that topological entities are returned in a predictable order. However, this is not the case with all geometry kernels. Siemens Parasolid, for example, does not return faces, edges and vertices in a predictable order [30]. The ordered naming method is also not possible if persistent naming is required across multiple different CAD systems. Jing et al. present methods for persistent naming in a replicated, multi-user CAD system across multiple different CAD systems. This technique is based on geometric properties of the object and are limited to simple geometry and swept features [14,15]. In a previous paper, we present a general method to identify topological entities based on unique feature and geometric properties [9].

3. LAZY NAMING TO ENHANCE FEATURE CREATION PERFORMANCE

The eager naming method identifies and names every face and edge immediately after a feature is created. This method of identifying topological entities is computationally expensive for models with many faces and edges. Conversely, lazy naming identifies and names topological entities when they are used by dependent features. Since the number of referenced entities is far less than the total number of entities that exist in the model, lazy naming saves a significant amount of computation time not having to identify nearly as many topological entities.

For example, when creating fillets on the top edges of a cube (as in Fig. 3), only the four edges that are referenced by the fillet operation are identified and named with the lazy naming method. Conversely, in the eager method, each edge and face in the cube are identified and named immediately after creation. In addition, after the fillets are applied, all the newly created edges and faces must be named. This eager process would require the identification and naming of 24 edges and 10 faces.

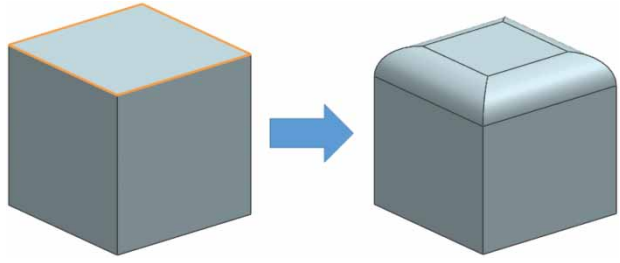


Fig. 3: The fillet operation with the eager method identifies 34 entities vs. the lazy method which requires identifying only 4.

The lazy naming method functions as follows:

1. An operation which references topological entities is performed by a client
2. The referenced topological entities of the operation are given unique names by the client
3. Unique geometric properties are found for the topological entities
4. The body on which the named topological entities reside are identified (see details below)
5. The unique name, geometric properties and body identifier are sent to the server along with the operation data
6. The server forwards the operation data, unique name, geometric properties and body identifier to all remote clients
7. The remote clients identify the topological entities by the body on which they reside and the unique geometric properties and give them the unique names
8. The remote client performs the operation referencing the topological entity by its newly given unique name

In order for the lazy naming method to uniquely identify faces and edges, the body on which these topological entities reside must be uniquely identified (as seen in step 4). The eager naming method identifies bodies by the edges and faces which comprise the body. This is not possible with the lazy naming method because not all the topological entities are identified. Instead, a body is uniquely identified by the feature which originally created it and its bounding box. Bodies can be uniquely identified by the feature which created it, except for in the case where a feature creates multiple bodies. When a feature creates multiple bodies, the bounding box is employed to uniquely identify each body created by the feature. Therefore

the combination of the creation feature and bounding box uniquely identifies all bodies in a CAD model, assuming features don't create multiple bodies in the same space.

4. DATA CACHE AND NORMALIZATION TO ENHANCE FEATURE EDIT PERFORMANCE

Central to the feature edit enhancement is the entity dictionary. There are two separate implementations of the entity dictionary, one on each client and one in the central database. The client entity dictionary is a cache which stores a relation between the persistent names of topological entities, their geometric properties, and computer memory pointers which reference those entities in the model. The database entity dictionary is the authoritative, persistent copy of the geometric properties and names of the topological entities. The topological entities are stored separately from the features that use them, allowing updates to entities to take place without having to update all the features which reference them.

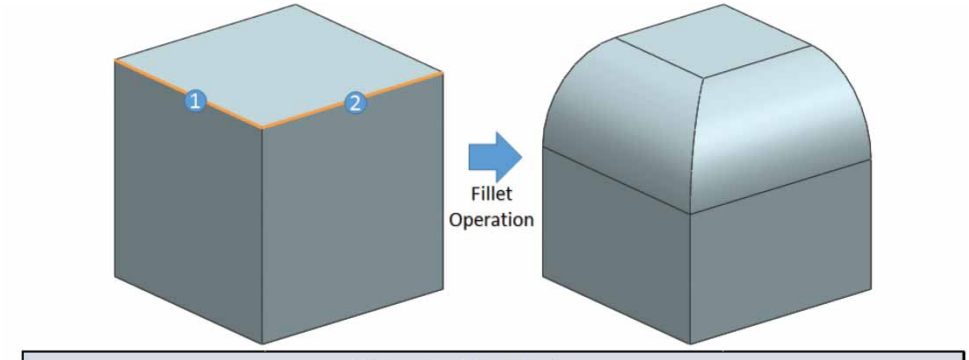
4.1. Client Entity Dictionary Cache

In a replicated, multi-user CAD environment, after an operation is performed, the data required to recreate that operation on other client machines is put into a data object. The data object is then forwarded to the server and distributed to other clients. As part of the lazy naming method, all topological entities referenced by the operation undergo the naming process. This process caches the following data in the client entity dictionary for easy access: (1) the unique name of the entity, (2) the geometric properties to uniquely identify the entity on a remote client and (3) the computer memory pointer to the entity on the local client. The first and second elements are included with the operation data sent to other clients when an operation is performed.

Fig. 4, illustrates the naming process that occurs after the user performs a fillet operation. The user selects edges 1 and 2 to perform the operation. After the operation is complete, entries are created in the client entity dictionary storing the entity name, Geometric ID (G-ID) and computer memory pointer. The G-ID holds unique geometric properties as well as the body of which the topological entity is attached.

When a remote operation is received from the server, the local entity dictionary is searched to make sure the referenced entities have been previously identified. If a topological entity name is not found in the remote client's dictionary, the geometric properties are used to find the topological entity on a given body and add it to the entity dictionary. This is done using a tolerance to compare the geometric properties of the G-ID of the entity to all of the geometric properties of the topological entities of the body it is attached to. When the topological entity is identified, it is assigned the same unique name forwarded from the remote client. The unique name and local memory pointer is cached in the client entity dictionary for quick look-up when another remote operation is received.

A critical aspect to this method is that feature operation data and topological entity data are isolated. The eager naming method stores topological entity data together with feature operation data. If this data is not normalized, whenever a user performs an edit operation, both the geometric properties of the topological entities and operation data required to recreate the dependent features of the edit operation must be extracted and resent to the server. By storing the entities separate from the features which use them, dependent feature data does not need to be extracted and resent for every edit operation. Only updated geometric data for all changed topological



Client Entity Dictionary

	Entity Name	Geometric ID	Entity Memory Pointer
1	dan_feature_3_edge_1	Edge[{(10,15,20)(5,15,20)(0,15,20)}Body_1]	0x5A884 [Edge 27]
2	dan_feature_3_edge_2	Edge[{(10,0,20)(10,7.5,20)(10,15,20)}Body_1]	0x03D20 [Edge 32]

Fig. 4: The client entity dictionary contains the entity name, Geometric ID and entity memory pointer for a specific client. The memory pointer is stored for quick access to the entity on a client.

entities in the model are sent to the server with the operation data. The updated geometric data is easily found by querying the client entity dictionary cache for changed entities.

4.2. Normalized Database Entity Dictionary

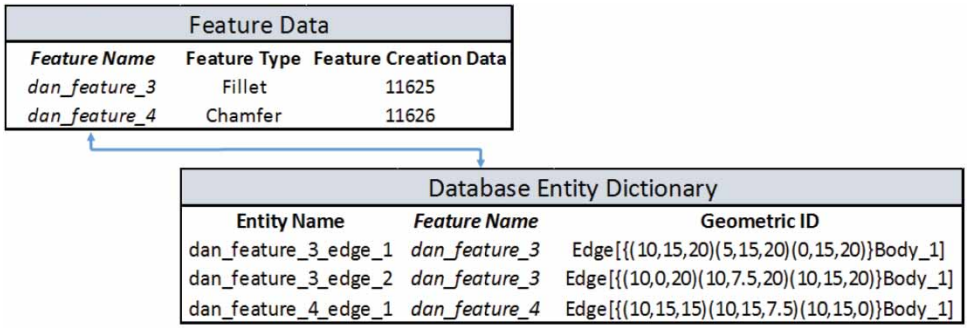
As features are created, edited, or deleted, data used to replicate these operations on other clients are forwarded through a server and stored persistently in a database. This data is the authoritative source of the model. It fully describes how to recreate the feature on another client and includes the geometric properties of topological entities that are referenced by the feature operation. As features are accepted by the server from remote clients, the data is added to the database to allow users who load the model in the future to download the most up-to-date features and topological entity data.

The database entity dictionary stores the topological entity data separate from the feature operation data, as seen in Fig. 5. The entity dictionary consists of the persistent name of the referenced topological entity, the unique geometric information for identification, and the name of the feature which references the entity. The database also stores the feature data which consists of the data necessary to create the feature on the client. The feature data references the entity data in the database so that when a feature is applied to a client the appropriate entity data is referenced.

By normalizing the data, any changes to the topological entity can be updated in the database entity dictionary without having to update the feature data. Edit and delete operations change the geometric properties of some topological entities. If a dependent feature references one of those changed entities, this data needs to be updated in the database. It is important to have the feature and entity data sets separated to avoid unnecessarily updating the feature data set which requires an update to the dependent features on all clients. This saves a time because all dependent features do not need to be updated when their parent feature changes.

4.3. Coordinating the Database and Client Entity Dictionaries

When a part is loaded, operations are performed based on the feature and entity data which are stored



Feature Data

Feature Name	Feature Type	Feature Creation Data
dan_feature_3	Fillet	11625
dan_feature_4	Chamfer	11626

Database Entity Dictionary

Entity Name	*Feature Name*	Geometric ID
dan_feature_3_edge_1	*dan_feature_3*	Edge[{(10,15,20)(5,15,20)(0,15,20)}Body_1]
dan_feature_3_edge_2	*dan_feature_3*	Edge[{(10,0,20)(10,7.5,20)(10,15,20)}Body_1]
dan_feature_4_edge_1	*dan_feature_4*	Edge[{(10,15,15)(10,15,7.5)(10,15,0)}Body_1]

Fig. 5: The feature data references the entity dictionary in the database. The database entity dictionary is the global reference for all clients.

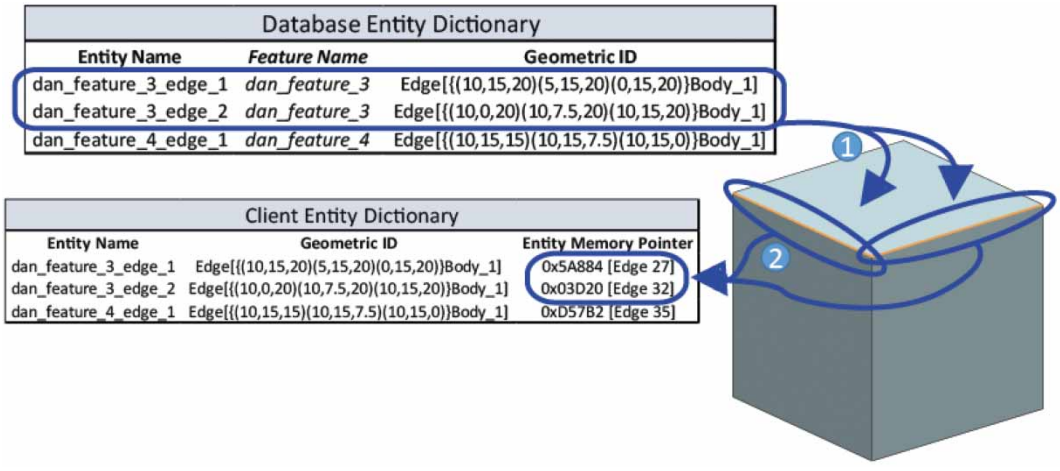


Fig. 6: Image showing the relationship between feature creation and the client entity dictionary.

in the database. As each operation is performed, a referenced entity is uniquely identified using the unique geometric properties of the entity. Once the entity is found, the local memory pointer, along with the entity name and G-ID are stored in the client entity dictionary. After all the referenced entities for the feature are stored in the client entity dictionary, the feature is created based on the data referenced from the client entity dictionary. Fig. 6 shows a representation of the database entity dictionary (upper table) which directly corresponds to topological entities on the client model. The lower table in Fig. 6 depicts the client entity dictionary.

When create or edit operations are performed on the client it is critical that all the new or updated data be sent to the server. The most recent geometric properties of the changed topological entities are compared against the geometric properties stored in the client entity dictionary to determine which entities have updated. When the most recent topological entity data is different or missing from that found in the client entity dictionary, the geometric properties of these entities are sent to the server with the edit or delete operation data. The operation data together with the updated topological entity data is termed an update message. Although the operation data and updated topological entity data is stored separately, it is critical that the update message contain both pieces of data because the entire message must be either accepted or rejected by the server as a single unit to ensure that it occurs as one operation. The server verification process for model consistency is outlined in [10].

5. ENHANCEMENT IMPLEMENTATIONS IN NXCONNECT

The lazy naming, data cache and normalization enhancements were implemented in NXConnect to improve feature creation and edit performance. The enhanced geometric property extraction was also implemented to additionally improve feature edit performance. NXConnect is a replicated multi-user CAD system under development at BYU. It has been developed as a plugin to Siemens NX and extends the functionality of the existing system to be multi-user using the NX Open API. The client-server architecture allows data to be sent between clients who each have a replicated instance of the model. As operations are performed by one client, the data to perform the operation is extracted by the plugin and sent to the server. The server forwards data to remote clients and has logic to ensure that operations on all clients occur in a consistent order [10,11, 22, 26–29, 36].

When an operation is performed on a client in which a topological entity is referenced, uniquely identifying attributes are extracted and placed into the client entity dictionary. The NX API uniquely identifies each topological entity with an identifier called a Journal ID (J-ID). A J-ID is a string value which contains an ephemeral name and geometric properties for an entity. The enhanced geometric property extraction utilizes the J-ID to rapidly extract geometric properties from an entity in a faster way than querying the entity for its geometric properties through the API as done in [9]. Alone, the J-ID is not necessarily unique. However, it becomes unique if the geometric properties of the J-ID are combined with a uniquely identified body, which is given a Body ID (B-ID). The B-ID is concatenated with the geometric data and put into the client entity dictionary as the G-ID. This G-ID is forwarded with the operation data to the server for verification and subsequent dissemination to other clients.

When an operation is verified by the server, the database entity dictionary is updated by adding or updating a row in the topological entity and feature tables in the SQL database on the server. At the same time, the operation and topological entity data is sent to remote clients. When an operation is received by a remote client, all unverified operations are suppressed to put the model in the same state as when the operation was performed on the originating client. Next, the client entity dictionary is searched by name to see if the topological entity had already been identified. If it is not found in the entity dictionary, the entity is identified in the model. The B-ID forwarded from the originating client is used to identify the body with the same corresponding B-ID in the remote client. Once the body is identified, the J-ID of each edge or face of that body is extracted and compared with the G-ID forwarded from the originating client. This identifies the corresponding face or edge which the operation should reference. The operation is then performed on the client.

When loading an existing part from the database, the client entity dictionary is initialized in the following way: A part file is stored in the database which is uploaded at checkpoints determined by the users, as described by [9]. When a checkpoint is set, a handle (persistent pointer) to each topological entity is stored with the entity name as a string in the attributes of the part. When the part is loaded, the string is parsed to extract the handles of each topological entity. This handle is used to get the memory pointer and unique name of each topological entity. This allows the client entity dictionary to be initialized without having to re-identify each topological entity, which saves a significant amount of time on load.

6. RESULTS

To validate that the three enhancements significantly improve times for feature creation and edits, several timed tests were performed to compare the implementation with these enhancements to the eager naming implementation presented in [9]. Several tests were run on three different implementations of

persistent naming methods in NXConnect. The first implementation tested is the eager naming implementation presented in [9]. The second implementation tested incorporates the lazy naming, data cache and normalization enhancements that can be applied to multiple CAD systems. The third implementation tested additionally includes the enhanced geometric property extraction that is only applicable to NX. Three runs for each implementation for several models are performed. The tests were performed on an Intel(R) Xeon(R) CPU E5–1603 0 at 2.80 GHz with 16 GB ram with 64-bit Windows 7.

6.1. Feature Creation Enhancement Tests

The first two tests are designed to test the feature creation enhancement of lazy naming. The first test was an extrusion of a rack for a rack and pinion assembly which creates 204 faces and 606 edges (shown in Fig. 7). The average computation time of three runs of the rack extrusion using the eager naming implementation was 1 min. 30 sec. on the originating client. Conversely, the average time it took for three tests using lazy naming with was 3 sec., representing a 30X speed-up. The enhanced geometric property extraction had no additional benefit for this creation operation because no geometric property extraction takes place when a create operation is performed (see Tab. 1).

Implementation	Ave. time of 3 tests	Speed-up
Eager naming	1 min. 30 sec.	
Lazy naming, data cache and normalization enhancements	3 sec.	30X
Plus enhanced geometric property extraction	3 sec.	30X

Tab. 1: Implementation time comparison for the extrusion of rack test.

The second test was the extrusion of a grate presented in the introduction and shown in Fig. 1. This operation creates 406 faces and 1212 edges. The average computation time of three runs of the extrusion operation and naming logic using the implementation in [9] was 4 min. and 39 sec. However, the average time it took using lazy naming was 6 sec. The remaining 6 sec. is the time for NX to perform the operation, extract the operation data and perform error rejection and consistency logic [10]. The pinion and grate tests show a 47X speed-up using the lazy naming method (see Tab. 2). Again, the enhanced geometric property extraction had no additional benefit because this is a creation operation.

Implementation	Ave. time of 3 tests	Speed-up
Eager naming	4 min. 39 sec.	
Lazy naming, data cache and normalization enhancements	6 sec.	47X
Plus enhanced geometric property extraction	6 sec.	47X

Tab. 2: Implementation time comparison for the extrusion of the grate.

6.2. Feature Edit Enhancement Tests

The next tests show the significant time savings with the feature creation and edit enhancements. Since there is a significant difference between the times it takes to perform the operation on the originating client compared to the time it takes to perform the operation on a remote client for the lazy naming implementations, both times are measured in these tests. Since the eager method performs the same logic on the originating client as on a remote client, only the originating time is reported.

The third test is a gear with a fillet operation creating fillets on the inside and outside edges of the teeth (see Fig. 8). Although the extrusion only has one dependent feature (the fillet operation), editing the extrusion requires the identification of 83 faces and 242 edges for the extrusion with an additional 80 faces and 240 edges for the fillet operation. This is a total of 163 faces and 482 faces to identify when the feature and dependent feature are identified using the eager implementation. The average computation time of three tests of the extrusion edit operation was

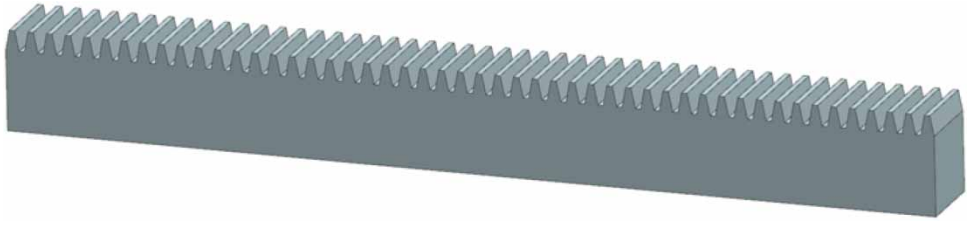


Fig. 7: Extrusion of rack creating 204 faces and 606 edges.

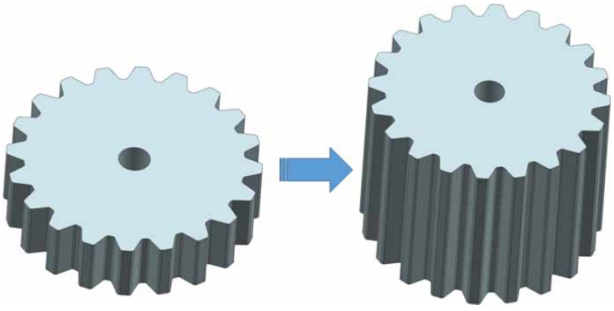


Fig. 8: Edit of extrusion length of gear with fillets.

41 sec. using the eager naming implementation. Conversely, it took 15 sec. to perform the edit operation with the lazy naming, data cache and normalization enhancements on an originating client and 7 sec. on a remote client. It only took 4 seconds to perform with the additional enhanced geometric property extraction on an originating client and only 1 sec. on a remote client (see Tab. 3).

The fourth test is the edit of a feature with several child components. This test is the editing of the extrusion height of a cylinder approximated by chamfering a cube several times. Fig. 9 shows how this approximated cylinder is created. The leftmost image in Fig. 9 shows the extrusion of a square. The middle image shows chamfers around the edges creating an octagon extrusion. Each of those edges are again chamfered. This process is repeated six times, creating an extrusion of a body with 256 sides which approximates a cylinder, shown in the far right image. The resulting cylinder approximation body has a total of 258 faces and 768 edges.

The test performed on this body is to edit the value of the original square extrusion to make it half of the original length (see Fig. 10). This operation requires a re-evaluation of all the dependent chamfers so they

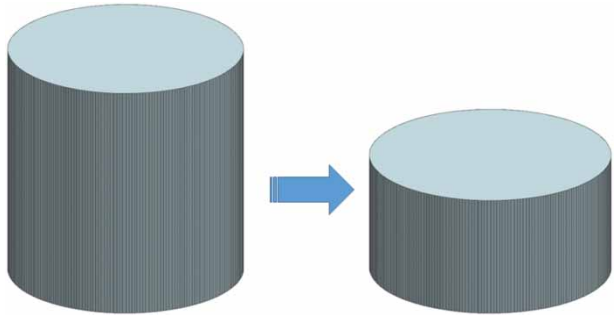


Fig. 10: Edit extrusion length to half the original length.

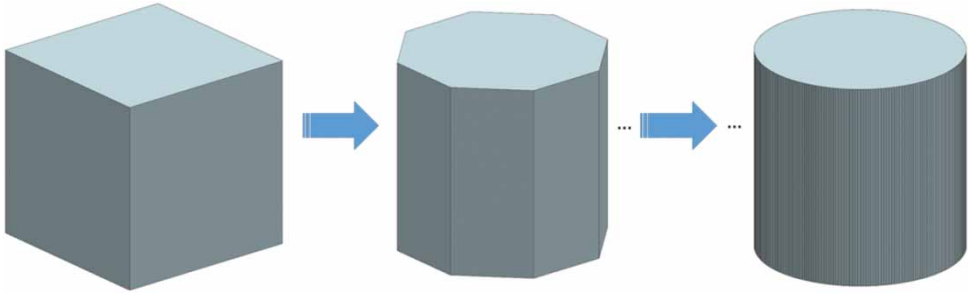


Fig. 9: Cube with many chamfers added until it approximates a cylinder with 256 sides.

Implementation	Ave. time of three tests	Speed-up
Eager naming	41 sec.	
Lazy naming, data cache and normalization enhancements (originating)	15 sec.	3X
Lazy naming, data cache and normalization enhancements (remote)	7 sec.	6X
Plus enhanced geometric property extraction (originating)	4 sec.	10X
Plus enhanced geometric property extraction (remote)	1 sec.	40X

Tab. 3: Implementation time comparison for the gear extrusion length edit.

Implementation	Ave. time of 3 tests	Speed-up
Eager naming	7 min. 35 sec.	
Lazy naming, data cache and normalization enhancements (originating)	1 min. 43 sec.	4X
Lazy naming, data cache and normalization enhancements (remote)	30 sec.	15X
Plus enhanced geometric property extraction (originating)	8 sec.	57X
Plus enhanced geometric property extraction (remote)	6 sec.	76X

Tab. 4: Implementation time comparison for the extrusion edit of the approximated cylinder.

update as well. The average of three tests using the eager naming implementation is 7 min. and 35 sec. With the eager implementation, the edit operation requires the identification of all the faces and edges after each chamfer operation, totally 522 faces and 1524 edges. The lazy naming, data cache and normalization enhancements only takes an average of 1 min. 43 sec. on an originating client and 30 sec. on a remote client. Conversely, the implementation with the enhanced geometric property extraction took only 8 sec. on an originating client and 6 sec. on a remote client. (see Tab. 4)

Significant time savings were seen with the enhancements in all four test cases. For the latter two tests, the time to perform the operation on the remote client was significantly less than the time it took on the originating client. The reason it takes more time on the originating client is because the originating client must determine which entities have moved in order to send the geometric properties of the updated topological entities to the server. Since the remote clients receive the updated geometric properties of the topological entities, they update the client entity dictionary directly, taking much less time.

The enhanced geometric property extraction implementation significantly improved times for edit operations. The speed-up is even more significant for feature edits with several dependencies. The geometric property extraction enhancement improves times for edit operations because it speeds up the geometric property extraction time in NX. Geometric property extraction is fundamental in determining which entities have moved after an edit operation. Conversely, no geometric property extraction takes place when a create operation is performed since the lazy naming approach delays naming of topological entities until they are referenced by a dependent feature.

7. CONCLUSION

We present three enhancements to dramatically reduce the time required for persistent naming in a multi-user CAD system compared with the eager naming method. The first enhancement improves feature creation performance using lazy naming which defers naming topological entities until they are actually referenced by dependent features. The second method enhancement improves feature edit performance by caching data on the client using an entity dictionary. It also normalizes the topological entity data and operation data to alleviate the need to update dependent features when an edit operation takes place. The third enhancement utilizes the NX Journal ID to rapidly extract geometric properties of topological entities to additionally enhance edit performance. Testing these enhancements in NXConnect shows that these speed-up the naming process in multi-user CAD by at least an order of magnitude over the eager naming method. Significant time savings benefits are also seen on feature edits with dependent features, especially for features with several dependencies. Time savings are even more significant for feature edits with the enhanced geometric property extraction implementation in NXConnect.

REFERENCES

[1] Baba-Ali, M.; Marcheix, D.; Skapin, X.: A method to improve matching process by shape characteristics in parametric systems, Computer-Aided Design and Applications, 6(3), 2009, 341–350. http://dx.doi.org/10.3722/cadaps.2009.341-350

[2] Bidarra, R.; Nyirenda, P.; Bronsvoort, W.: A feature-based solution to the persistent naming problem, Computer-Aided Design and Applications, 2(1), 2005, 517–526. http://dx.doi.org/10.1080/16864360.2005.10738401

[3] Bidarra, R.; Bronsvoort, W.: Persistent naming through persistent entities, Geometric Modeling and Processing, Proceedings, 2002, 233–240.

[4] Bidarra, R.; Van den Berg, E.; Bronsvoort, W. F.: A Collaborative Feature Modeling

System, Journal of Computing and Information Science in Engineering, 2(3), 2002, 192. http://dx.doi.org/10.1115/1.1521435

[5] Capoyleas, V.; Chen, X.; Hoffmann, C.: Generic naming in generative, constraint-based design, Computer-Aided Design, 28(1), 1996, 17–26. http://dx.doi.org/10.1016/0010-4485(95)00014-3

[6] Chen, X.; Hoffmann, C.: On editability of feature-based design, Computer-Aided Design, 27(12), 1995, 905–914. http://dx.doi.org/10.1016/0010-4485(95)00013-5

[7] Chen, Z.; Gao, S.; Zhang, F.; Peng, Q.: An approach to naming and identifying topological entities, Chinese Journal of Computers, 24(11), 2001, 1170–1177.

[8] Dietrich, U., v. Lukas, U., Morche, I.: Cooperative modeling with TOBACO, Proceedings of TeamCAD: GVU/NIST Worskshop on Collaboritive Design, 1997.

[9] Hepworth, A.; Nysetvold, T; Bennett, J.; Phelps, G.; Jensen, C. G.: Scalable Integration of Commercial File Types in Multi-User CAD, Computer-Aided Design & Applications, 11(4), 2014, 459–467. http://dx.doi.org/10.1080/16864360.2014.881190

[10] Hepworth, A. I.; Tew, K.; Trent, M.; Ricks, D.; Jensen, C. G.; Red, W. E.; Model Consistency and Conflict Resolution with Data Preservation in Multi-user CAD, Journal of Computing and Information Science in Engineering, accepted 2014. http://dx.doi.org/10.1115/1.4026553

[11] Hepworth, A. I.; Tew, K.; Nysetvold, T.; Bennett, M.; Jensen, C. G.; Automated Conflict Avoidance in Multi-User CAD, Computer-Aided Design & Applications, 11(2), 2014, 141–152. http://dx.doi.org/10.1080/16864360.2014.846070

[12] Jing, S.; He, F.; Han, S.; Cai, X.; Liu, H.: A method for topological entity correspondence in a replicated collaborative CAD system, Computers in Industry, 60(7), 2009, 467–475. http://dx.doi.org/10.1016/j.compind.2009.02.005

[13] Jing, S.; He, F.; Cai, X.; Liu, H.: Collaborative naming for replicated collaborative solid modeling system; International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, 2008, 141–150.

[14] Jing, S.; Yuan, Q.: Consistent Naming for Sweeping Features in Replicated Collaborative Modeling System, International Conference on Information Management, Innovation Management and Industrial Engineering, 2009, 90–93.

[15] Jing, S.; Yuan, Q.: Consistent Naming for Sweeping Features in Replicated Collaborative Modeling System, Computer-aided Industrial Design and Conceptual Design, 2009, 899–904.

[16] Kao, Y.; Lin G.: Development of a collaborative CAD/CAM system, Robotics and Computer-Integrated Manufacturing, 14(1), 1998, 55–68. http://dx.doi.org/10.1016/S0736-5845(97)00014-8

[17] Kripac, J.: A mechanism for persistently naming topological entities in history-based parametric solid models, ACM Symposium on Solid Modeling and Applications, 3, 1995, 21–30. http://dx.doi.org/10.1145/218013.218024

[18] Liao, B.; He, F.; Jing, S.: Replicated Collaborative Solid Modeling and Naming Problems, Ninth International Conference on Computer Aided Design and Computer Graphics, 2005.

[19] Liao, B.; He, F.; Jing, S.; Wu, Y.: A Transformation-Based Method for Name Converge in Quiescent Context of Replicated Solid Modeling Systems, Proceedings of the 10th International Conference on Computer Supported Cooperative Work in Design, 2006.

[20] Marcheix, D.; Pierra, G.: A survey of the persistent naming problem, Proceedings of the seventh ACM symposium on Solid modeling and applications, 2002, 13–22. http://dx.doi.org/10.1145/566282.566288

[21] Mishra, P.; Varshney, A.; Kaufman, A.: CollabCAD: A Toolkit for Integrated Synchronous and Asynchronous Sharing of CAD Applications, In J. Rossignac (Ed.), Proceedings TeamCAD: GVU/NIST workshop on collaborative design, 1997, 131–137.

[22] Moncur, R.; Jensen, C.; Teng, C.; Red, E.: Data Consistency and Conflict Avoidance in a Multi-User CAx Environment, Computer-Aided Design and Applications, 10(5), 2013, 727–744. http://dx.doi.org/10.3722/cadaps.2013.727-744

[23] Nam, T.; Wright, D.; Collide: A Shared 3D Workspace for CAD, Proceedings of Conference on Network Entities, 1998.

[24] Qiang, L.; Zhang, Y. F.; Nee. A.Y.C.: A Distributive and Collaborative Concurrent Product Design System through the WWW/Internet, The International Journal of Advanced Manufacturing Technology, 17(5), 2001, 315–322. http://dx.doi.org/10.1007/s001700170165

[25] Ramani, K.; Agrawal, A.; Babu, M.; Hoffmann, C.: CADDAC: Multi-Client Collaborative Shape Design System with Server-based Geometry Kernel, Journal of Computing and Information Science in Engineering, 3(2), 2003, 170. http://dx.doi.org/10.1115/1.1582882

[26] Red, E.; Jensen, C.; Holyoak, V.; Marshall, F.; Xu, Y.: v-Cax: A Research Agenda for Collaborative Computer-Aided Applications, Computer-Aided Design and Applications, 7(3), 2010, 387–404. http://dx.doi.org/10.3722/cadaps.2010.387-404

[27] Red, E.; Jensen, C.; French, D.; Weerakoon, P.: Multi-User Architectures for Computer-Aided Engineering Collaboration, International Conference on Concurrent Enterprising, 2011.
[28] Red, E.; French, D.; Jensen, G.; Walker, S.; Madsen, P.: Emerging Design Methods and Tools in Collaborative Product Development, Journal of Computing and Information Science in Engineering, 13(3), 2013, 1–13. http://dx.doi.org/10.1115/1.4023917
[29] Red, E.; Jensen, C. G.; Weerakoon, P.; French, D.; Benzley, S.; Merkley, K.: Architectural Limitations in Multi-User Computer-Aided Engineering Applications, Computer and Information Science, 6(4), 2013, 1–16. http://dx.doi.org/10.5539/cis.v6n4p1
[30] Siemens Corp., Parasolid Documentation.
[31] Stork, A.; Jasnoch, U.: A Collaborative Engineering Environment, Proceedings of the TeamCAD97 Workshop on Collaborative Design, 1997, 25–33.
[32] Stork, A.; Lukas, U.; Schultz, R.: Enhancing a Commercial 3D CAD System by CSCW Functionality for Enabling Co-operative Modelling via WAN, Proceedings of the ASME Design Engineering Technical Conferences, 1998.
[33] Tang, M.; Chou, S. C.; Dong, J. X.: Conflicts classification and solving for collaborative feature modeling, Advanced Engineering Informatics, 21(2), 2007, 211–219. http://dx.doi.org/10.1016/j.aei.2006.05.006
[34] Wang, Y.; Nnaji, B.O.: Geometry-based semantic id for persistent and interoperable reference in feature-based parametric modeling, Computer Aided Design, 37(10), 1081–1093, 2005. http://dx.doi.org/10.1016/j.cad.2004.11.009
[35] Wu, J.; Zhang, T.; Zhang, X.; Zhou, J.: A face based mechanism for naming, recording and retrieving topological entities, Computer-Aided Design, 33(10), 2001, 687–698. http://dx.doi.org/10.1016/S0010-4485(00)00099-3
[36] Xu, Y., Red, E., Jensen, C.: A Flexible Context Architecture for a Multi-User GUI, Computer-Aided Design & Applications, 8(4), 2011, 479–497. http://dx.doi.org/10.3722/cadaps.2011.479-497
[37] Zhou, X. and J. Li: A Web-based synchronized collaborative solid modeling system, Chinese Journal of Computer Integrated Manufacturing Systems, 2003, 960–965.
[38] Zhou, X.; Gao, S.; Li, J.; He, F.: Flexible concurrency control for synchronized collaborative design, Proceedings of 2003 ASME DETC/CIE Conference, 1, 2003, 591–598.