A Selective Undo/Redo Method in 3D Collaborative Modeling Environment

Yuan Cheng
School of Computer Science
Wuhan University
Wuhan 430072 P.R.China
graphics@whu.edu.cn

Fazhi He
School of Computer Science
Wuhan University
graphics@whu.edu.cn

Xiantao Cai
School of Computer Science
Wuhan University
graphics@whu.edu.cn

Dejun Zhang
School of Computer Science
Wuhan University
graphics@whu.edu.cn

***Abstract*—In 3D collaborative modeling systems, users need a convenient mechanism to repeatedly modify the models they are operating on. In this paper, we contribute a selective undo/redo solution for users to select arbitrary operation to undo. With the consistency maintainence mechanism we proposed, operations need to be re-arranged on each site for after their arriving. Both history buffer and model state stream are adopted to present the arriving sequence of operations and their actual execution sequence. In case of concurrent undo/redo, undo state vector is proposed to make sure that an operation can only be undone once and redone by the designer who undoes it. Based on all the precautions we have made, an undo/redo algorithm is proposed. The algorithm has been verified in the prototype we implemented.**

Keywords-3D collaborative modeling; Model State Stream; Concurrent undo/redo;

I. INTRODUCTION

Human plays an instructive role in interactive systems. From the perspective of collaborative CAD systems, the ultimate designing result is the embodiment of mutual intention of clients and group designers. Objectively speaking, it is inevitable that designers may make mistakes during the designing process. Simple slips and lapses at this level account for roughly 60% of human errors. The highest cognitive level is knowledge-based, where tasks are approached by reasoning from first principles, without the aid of previously-formed rules or skills; mistakes at this level account for the 10% of human errors[1]. All in all, during the process of searching for optimal solution, designers need to undo some of the executed operations to roll back the document state and then execute a series of new operations to try for different solutions.

Being a widely used error recovery mechanism, Undo/Redo can fully meet the requirements of error recovery and optimal solution exploration in interactive systems. It is more indispensable in collaborative editing systems. In such systems, data are replicated on each site so designers from different site are given the chance to edit each replica both on-line and off-line. During the on-line collaboration, to share the designing result, operations from local and remote sites are interleaved arbitrarily due to concurrency and it brings more challenges to Undo/Redo function.

Based on the basic multi-user undo/redo requirements Dewan and Choundary[2] proposed, undo/redo mechanism in 3D collaborative modeling systems should have the following criteria:

- Dual Undo/Redo mechanism. Undo/redo mechanism in multi-user systems should have the same characteristic that single user interactive systems have. When editing the replica on a local site, a designer should be freely to undo/redo any operation been issued by himself. That is local undo/redo mechanism. Meanwhile, a user should also be entitled to undo the last operation ever executed which is probably not the last operation he himself issued. That is in accordance with the global undo/redo mechanism.
- Atomicity. The execution of an advanced modeling operation is actually composed of several sub-operations. Set CUBIC_PROTRUSION_ATTACHMENT as an example. First, a cubic block is created according to dimension parameters. Second, the block is translated and rotated to properly locate on the specific face of the base model. Third, the Boolean Union operation is called and the boundary model is re-evaluated. However, when a user undoes O, all these sub-operations should be treated as integrity. Effects of all these operations should be eliminated.
- Fast Response. When a user issues an operation, no matter it is a normal do or undo, the result is expected to be displayed as fast as possible. 3D model construction is the key issue that effects. More modeling operations means slower response.

978-1-4673-1212-7/12/$31.00 ©2012 IEEE

- Selective Undo/Redo. In local undo/redo mechanism, the last operation local user issued is not the last operation executed on other remote sites. In global undo/redo mode, the last operation executed on a local site is probably not the last operation executed on other sites as well. Therefore, selective undo/redo is the very critical requirement an undo/redo solution should satisfy. It is also a mainstream of the existing undo/redo model [3, 4, 5, 6, 7, 9 ,10,11].

In this paper, we introduce a selective undo/redo mechanism that fulfills the above requirements:

1) Two history views are presented for users to choose. Local Operation View is for designers to view and select arbitrary operation issued by himself and undo. The other is Full Operation View to present all the operations executed. Any operation can be selected as user wish. Actually, in history view operation arrangement is consistent with the operation arriving sequence on the site.

2) Each site keeps a Model State stream. The states of a model can be traversed along the stream.

3) Dependency among operations are analyzed, so the undo semantic can be satisfied.

The rest of this paper is organized as follows. In Section 2, the lated work is reviewed. Section 3 gives a brief description of the consistency model we adopted. Section 4 analysis the history buffer and model state stream kept on collaborative sites. Section 5 gives a detailed description of the undo/redo algorithm we proposed. Section 6 is the conclusion of our work.

II. Related Work

The initial researches of Undo/Redo model are in single-user environment. In multiuser collaboration systems, editing objects involve data records, texts, 2D graphics and bitmaps while no articles or prototypes concerning collaborative CAD systems are contributed. Abowd [9], Prakash [10], Berlage [3] classified undo models into 3 categories: 1) Single-step undo. 2) Linear undo. 3) History undo.4) Selective undo.

OT-based algorithms play an important role in solving undo problems where the creation,execution and integration ofinverse operation are critical. The typical algorithms include[4][5][7][8][10][11][13].

Prakash[10] proposed the transformation -based undo algorithm for the first time.The undo target O is transposed with the later executed operations to become O'.Then an inverse operation of O' is created and executed to cancel the O's effect. The undo process can be somehow simplified by taking some transpose reducing actions.

In Ressel's work [11] the undo mechanism is somehow different. First, a rough inverse version of the undo target in calculated and transformed against the later executed operations by the same user. All operations between the operation to be undone and its inverse should be undone first.

Sun's Anyundo [4][5] has the similar idea with[11]. Its basic idea of undo in is to treat Undo(O) as an inverse operation O' generated immediately after O but concurrently with all other operations. Redo can be easily implemented due to a do-undo pair and undo mark.

Both algorithms from Rellsel[11] and Sun[4][5] have deficiencies even they are seemingly to avoid conflicts effectively since they have confined user operations into a very limited area. In reality, group users are presented with multiple editing alterations other than merely INSERTION and DELETION such as changing the color or size of characters. For example, set the initial document state S as "abc". User X issues an operation O1 = Ins(2, "X"). The operation is executed on local site and then propagated to other remote sites. Thus, document states on both sites are turning into "aXbc". Then, user Y issues O2 to change the color of X. Both adOPTed and ANYUNDO didn't take situation like this into consideration.

Bin Shao and Du Li[7][8] integrates do and undo in one ABTU algorithm which can drastically improve the time complexity of undo.

However, the OT idea is not suitable for 3D environment since the modeling operations involved are far beyond two types like DELETION and INSERTION in textual environment. Their inversions are hard to obtain. Operations such as filleting, blending etc., may not have inverse operations.

III. Consistency Preservation in 3D Collaborative Modeling System

A. General Undo Principles

The undo/redo algorithm proposed in our paper is based on the following principles:

1) An undo operation is a meta-command that differs from the normal do operation. When an undo is executed, it is not put into the history buffer assembly with other do operations. To eliminate the effects of an undo, users can issue a corresponding REDO command which has the inverse effects with undo.

2) An operation can only be undone once. Even multiple users may aim at the same operation as their undo object, only one undo can be honored. Also an operation can only be redone by designer who undoes it.

3) Given any operation O, if there are operations depend on its effect, these operations should be undone as well when O is undone. In 3D collaborative modeling environment, creation and execution of an operation should refer to topological entities from the current document state. For example, O_1 creates a Base Block and O_2 adds a cubic protrusion on it. First, a boundary face F of the block should be designated. Second, distances to two orthogonal edges of face F should be given to fix this

protrusion on F. From this point of view, we can say that O_2 depends on O_1. If O_1 is undone, O_2 is meaningless and should be undone as well. However, when an operation is redone, operations depending on it can not be redone simultaneously.

B. Consistency Model

In collaborative editing systems, there are three properties that should not be violated: convergence, casual preservation and user intention preservation[12]. In a word, the convergence property guarantees the consistency of the final results, the causality-preservation guarantees the consistency of the execution order of the dependent operations, and the intention - preservation guarantees the consistency of the execution effects of independent operations [13]. Whenever a user sends an undo operation, document states on all sites should be the same after executing the undo.

Operation concurrency takes the responsibility for nearly all inconsistency problems. The fundamental pre-condition of a modeling operation execution is topological entity correspondence. Topological entities referenced in a local operation can be changed by concurrent operations when it is to be executed on any remote site. Either the operation cannot be executed or the execution of the operation may lead to model inconsistency. Many topological entity correspondence methods have been proposed while some of them can only be used in single-user environment. Here, we introduce a tree-like structure called TEST (Topological Entity Structure Tree) to record the changing history of each original topological entity from the initial boundary model on each site. Using TEST, whether a topological entity is obliterated, split or merged into other topological entities can be clearly recorded. With the consideration that each set of topological entities are created by a specific operation, when the relationship among topological entities are clarified, an operation's effecting operations can therefore be obtained.

Operations need to be re-arranged for its proper execution and model consistency on all sites to achieve causality preservation and intention preservation. Apparently, an operation's arriving sequence is not the same with its actual execution sequence of all operations due to operation re-arrangement in model state construction.

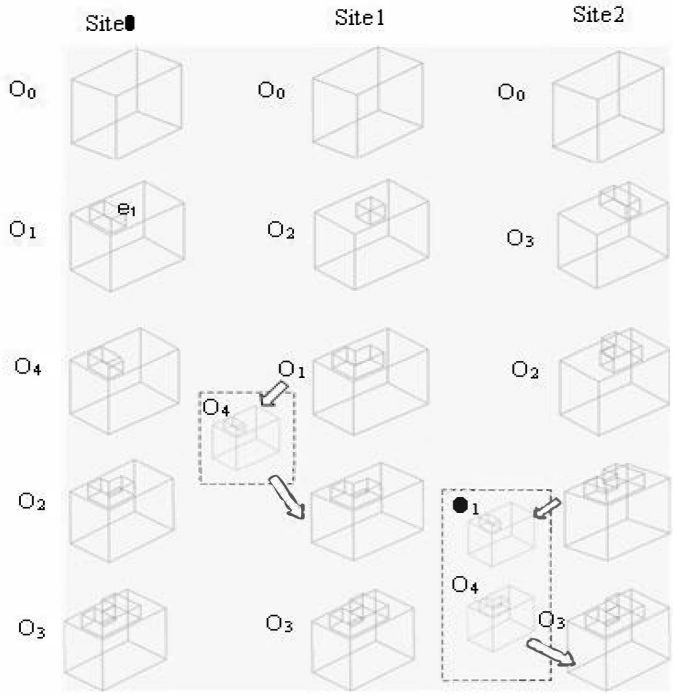


Figure 1. Operation Re-arrangement in Collaborative Modeling

In Figure.1, O_0 is issued on $site_0$ to create a cubic block BLOCK1. O_0 is then sent to $Site_1$ and $Site_2$. O_1 is created on site0 to create a cubic protrusion and its concurrent operations O_2 and O_3 are created on site1 and $site_2$ perspectively to also create cubic protrusions but with different locations. Then, O_4 is created on $site_0$ choosing e_1 to fillet. O_1 is then sent to the other two sites.

On $site_1$, the arriving sequence of all operations are: O_0,O_2,O_1,O_4,O_3. When O_4 arrives, the operation target e1 has merged with edge from the cubic protrusion created by O_2. To guarantee the correct execution of O_4, the operations need to be re-arranged to make sure e_1 can be precisely located. Thus, the execution sequence of operations on $site_1$ is : O_0,O_1,O_4,O_2,O_3.

On $site_2$, the arriving sequence of all operations are: O_0,O_3,O_2,O_1,O_4. When O_4 arrives, the operation target e_1 has merged with edges from the cubic protrusions created by O_2 and O_3. To guarantee the correct execution of O_4, the operations need to be re-arranged to make sure e_1 can be precisely located. Thus, the execution sequence of operations on $site_2$ is : O_0,O_1,O_4,O_2,O_3.

C. Undo State Vector

In UNDO/REDO mode, there can be the following situations: 1)An operation is undone;2) An operation is redone;3) An operation is undone more than once; 4) An operation is redone more than once; 5)An operation is the undo target of more than one site, we call this concurrent undo. In a collaborative modeling system with N sites, each site has a SiteID ranges from

1 to N. An Undo State Vector is an N tuple. Each element is initialized to 0. If an operation O is undone by the i*th* site, the i*th* element USV[i] is increased by 1. If O is redone by the i*th* site, the corresponding element USV[i] is decreased by 1.To any specific operation O, the following conclusions can be easily driven:

1) Undo state vector can be used to illustrate how many sites have aimed O as their undo target concurrently. In case of concurrent undo scenario, the first arrived undo will be processed and undo coming afterwards will not be processed but the corresponding element in USV is increased by 1.

2) If all elements of O's undo vector are 0, it means O is still in effects. It is either redone or never been undone.

3) The possible values for each element are 0 and 1.Whether an operation is undone or redone can be clearly described in the undo vector.

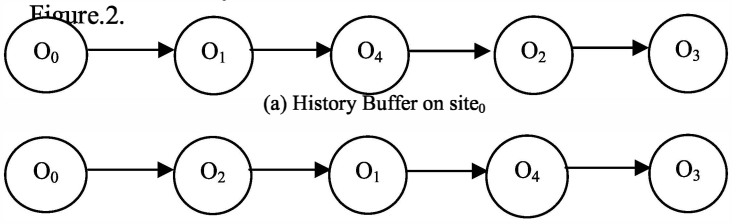
IV. HISTORY BUFFER AND MODEL STATE STREAM

In 3D collaborative modeling systems, operations are encapsulated, transmitted and stored in the form of advanced modeling commands which can clearly describe its origination:

O(SiteID, Creation_SEQ, OpID, Ref_Entity_List, ParaList)

- SiteID is the id of the site that creates O,
- Creation_SEQ is O's sequence number in all operations issued by the site with the same SiteID.
- OpID specifies the type of O. It is in variety since the primitive operations in 3D modeling systems diverse.
- Ref_Entity_List lists all the topological entities referenced by O.
- ParaList is all the parameters supporting the O's execution, eg. dimension parameters, location parameters.

An operation is put into history buffer as soon as it is executed. A history buffer is used to keep all the executed operations on a collaborative site. It can be browsed with Full Operation View. However, if we enter the Local Operation View, only part of the history buffer can be presented. This can be implemented by comparing the SiteID contained in an operation command and the local site id. Only operations owning the same SiteID with the site that requires Local Operation View can be displayed in Local Operation View. In example from Figure 1 above, the history buffers on $site_0$ to $site_2$ are described in Figure.2.



(a) History Buffer on $site_0$

(b) History Buffer on $site_1$

(c) History Buffer on $site_2$

Figure 2. History Buffer on site0 and site1 in collaborative modeling process

The execution of a modeling operation on a specific site i can change its model state into the next stage. This process can be described by the equation:

$$ModelState(i) + Op = ModelState(i+1)$$

With the continuous execution of modeling operations, the model state evolution can be described by Model State Stream. A model State Stream indicates the actual execution sequence of operations on a specific site due to operation re-arrangement. Each node in the stream is given an ID which is an integer between 1 and N. If there are N operations executed on a site, there must be at least N nodes in its model stream. Each operation in the history buffer should have a corresponding node delegating its execution state. Node(i) is the model state after executing the i*th* operation. Within a state stream, we can easily jump to any previous state by calling node(i), i = 1…N. However, the i*th* operation in history buffer isn't necessary the i*th* node in state stream due to the operation re-arrangement used in our consistency maintainence model. Take the example used in Fig.1, on $site_2$, the 2nd operation in its history buffer is O3 while O3 is the last node in model state stream. Figure.3 is the comparison of history buffer and model state stream on site2.

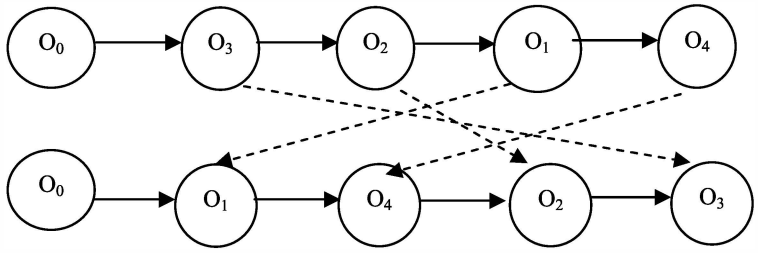


Figure 3. Linkage of history buffer and state stream on $site_2$

Therefore, there must be a linkage between an operation and the corresponding node in the model state stream. Whenever an operation arrives at a site, the arriving sequence ARR_SEQ is recorded. After consistency preservation and operation re-arrangement, the actual position of the operation during the model reconstruction process is recorded, this actual position is exactly the position of the corresponding state node in the model state stream, noted as NODE_SEQ.

A. Dependency Analysis

Operation dependency, also called conflicts in some literatures, is a must-be-considered factor during undo process, especially in 3D collaborative system. Details of our dependency detection algorithm has been illustrated in our CSCWD'09[6]

paper. So, it can be demonstrated that operations depend on the undo target O are executed after it, and the corresponding state node is behind O's state node. We add a DepFlag in O's corresponding command to represent which operation O depends on. When an operation is undone, any operation depending on it should be undone as well for the topological entities it refers do not exist anymore and their existence is meaningless. To represent if these operations are still in effect, a status symbol OpStat is also included, *ON* indicates the operation is not undone, *OFF* indicates the operation is undone because because of some dependency relationship. Whether an operation can be presented in the history view is determined by its status.

Eventually, an operation is encapsulated and stored in the history buffer in the following form:

O(SiteID, Creation_SEQ, OpID, Ref_Entity_List, ParaList, ARR_SEQ, NODE_SEQ, DepFlag, OpStat)

V. OUR UNDO/REDO ALGORITHM

A. Implementation of Undo on Local Site

When a local site selects an operation O to undo, it is executed on the local site immediately. During this course, s/he must be using one of the two history views and makes the choice. Undoubtedly, it is only when the undo target is available and selectable can it be chosen by the user. We use an UndoList to keep all the undone operations one by one. The process is described in Algorithm1.

If an undo is submitted on local site, the original operation is firstly checked to see whether it has already been undone. If so, this undo request will be aborted. Otherwise, we will roll back the model state back to the state where the operation before O is executed, get its dependency set, advance USV and re-execute effective operations right after O.

After an undo is processed on the local site, it is send to other sites in the form of UNDO(Undo_SiteID,Create_SiteID,Create_SEQ), Create_SiteID indicates the site that creates the undo target O and Create_SEQ is O's sequence number of total operations created by Create_SiteID.

```
Input: local site ID i, HB_i, local site model state
       stream MSV_i
Output: Re-evaluated geometry model
1:  Get the undo target O;
2:  if(O.OpStat == OFF) then exit;    //undo request
invalid
3:  else
4:    NodePos = O.NODE_SEQ;
5:  O.OpStat = OFF;
6:  CurrentState = MSV_i[Nodepos-1] ;
7:  DependencySet(O) = DependencyDetection(O);
8:  for( each operation in DependencySet(O)) do
9:    DependencySet[i].DepFlag = O;
10:   DependencySet[i].OpStat = OFF;
11: Endfor
12: for( each operation in HB_i) do
13:   if(HB_i[i].ARR_SEQ<O.ARR_SEQ
14:      and HB_i[i].NODE_SEQ > O.NODE_SEQ )
15:      do HB_i[i];
16:   endif
17:   if(HB_i[i].ARR_SEQ> O.ARR_SEQ and
18:       HB_i[i].NODE_SEQ > O.NODE_SEQ and
19:          is not in DependencySet(O))
20:        do HB_i[i];
21:   endif
22: Endfor
23: Reset MSV_i;
24: Adjust NODE_SEQ of each operation in HB_i;
25: for( each operation in DependencySet(O)) do
26:        DependencySet[i].USV[i] = 1;
27: Endfor
28: O is put in the UndoList_i;
29: endif
30: O.USV[i] = 1;
```

Algorithm 1. Undo on Local Site

B. Implementation of Undo on Remote Site

When some site j receives a remote undo command, it is executed following Algorithm 2.

```
Input:undo command, HBj on site j, model state
stream MSVj
Output: Re-evaluated geometry model
1: while( not the end of HBj)
2:    if(undo.Create_SiteID == HBj[i].SiteID
3:      && undo.Create_SEQ == HBj[i].CreateSEQ)
4:            then O = HBj[i];
5:    break;
6:   endif
7: endwhile
8: for(each element in O.USV)
9:   if(O.USV[m] == 1) //it means O is undone by
10:                          a concurrent undo
11:     O.USV[SiteID of the issuing undo site]==1;
12:     the undo command is discarded and
13:                  undo process on site_j is over;
14:  endif
15: endfor
16: NodePos = O.NODE_SEQ;
17: O.OpStat = OFF;
18: CurrentState = state[nodepos-1] ;
19: DependencySet(O) = DependencyDetection(O);
20: for( each operation in DependencySet(O)) do
21:           DependencySet[i].DepFlag = O;
22:           DependencySet[i].OpStat = OFF;
```

```
23: Endfor
24: for( each operation in HB_j) do
25:   if(HB_i[i].ARR_SEQ > O.NODE_SEQ and
26:     HB_i[i].ARR_SEQ<=O.ARR_SEQ) then
27:         do HB_j[i];
28:   endif
29: if(HB_j[i].ARR_SEQ is larger than O.ARR_SEQ
30:         and is not in DependencySet(O))
31:
32:          do HB_j[i];
33:   endif
34: endfor
35: Reset MSV_j;
36:Adjust NODE_SEQ of each on operation in HB_j ;
37: O is put in the UndoList_j;
38: for( each operation in DependencySet(O)) do
39:     DependencySet[i].USV[Undo_SiteID] = 1;
40: Endfor
41: O.USV[Undo_SiteID] = 1;
```

Algorithm 2. Undo on Remote Site

After a site receives an undo from a remote site, we first scan HB_j to find the original operation $HB_j[i]$ this undo aims at. Then, we identify whether $HB_j[i]$ has been undone by concurrent operation or due to operation dependency by checking if there exists an element in $HB_j[i]$.SUV equals to 1.If the undo target has already been undone, the undo process will be terminated.

Nevertheless, it's dependency set is obtained. Document state is rolled back to the state where the last operation right before $HB_j[i]$ leads to.Eventually, re-execute effective operations right after O.

C. *Implementation of Redo*

A user can redo when he intends to put the undone commands in effects again. To redo is to redo the latest operation ever been canceled. This can be realized by executing the redo target operation on current model state. However, only the site issues undo command is allowed to issue redo. Algorithm 3 gives the detailed description of the redo on local site.

```
Input: local site ID i, local site history buffer HB_i, local site
       model state stream MSV_i, UndoList_i kept on site i
Output: Re-evaluated geometry model
1: if(Local_Operation_View) then
2: O = the last operation in UndoList_i with its SiteID equals to
3:      i;
4: Execute O on current state;
5: A copy of O is added to the end of HB_i;
6: O.OpStat = ON;
7: O.ARR_SEQ = current number of ON nodes in HB_i;
8: O.NODE_SEQ = current number of state nodes in MSV_i;
9: All elements in O.USV are set to 0;
10: else // if it is in the Full_Operation_View
11:   O = the last operation in UndoList_i;
12:  Execute O on current state;
13:   A copy of O is added to the end of HB_i;
14:   O.OpStat = ON;
15:  O.ARR_SEQ = current number of ON nodes in HB_i;
16:  O.NODE_SEQ = current number of state nodes in 17:
17:  MSV_i;
16:   All elements in O.USV are set to 0;
17：endif
18：O is removed from UndoList_i;
```

Algorithm 2. Redo on Local Site

A redo command is then sent in the form of Redo(Create_Site, Create_SEQ). When a redo is sent to some remote site j, there can be the situation that different sites has aimed this same operation as the undo target concurrently. So, this redo requirement cannot be processed on all sites since its original undo requirement was not be processed. Just suppose a collaborative environment with 4 sites and N operations are issued. Both site1 and site3 aim at O_i as the undo target simultaneously. Afterwards, the undo from site1 is sent to site2 before site3 and undo from site3 is sent to site4 ahead of site1. When site1 redoes O_i, the requirement cannot be accepted by site3 and site4. This problem can be resolved by group negotiation.

If a user sends a redo command, the command should not be processed by the local site immediately. The redo requirement is sent to rest of the remote sites for them to choose whether to redo the target operation or not. Only when all group users agree to redo, the redo requirement can be processed both locally and remotely.

VI. CORRECTNESS PROOF

- Our undo/redo algorithm satisfies execution relationship preservation.This is easy to understand. During the undo/redo process, operation sequence is not sabotaged. Therefore, this process doesn't violate the execution relationship preserved order established by operation re-arrangement based on TEST.
- With our undo algorithm, convergence property is followed.The undo purpose is to eliminate the effect of the chosen operation. If there is only one undo at one moment, all sites will take the same action to undo the chosen operation and operations depend on it. If there is concurrent undo at one time, one on
- With our undo algorithm, user intention is preserved. The assembly line of undo target location, operation analysis, dependency set obtaining and document state reversal and new document state creation can exactly fulfill any user's undo requirement.

VII. Implementation and Test Results

To demonstrate the feasibility and effectiveness of the undo/redo method we proposed, we have make several experiments. The experiments involve local Undo/Redo preserving users' intention and correlativity processing among operations The process is shown in Fig. 2. The whole process involved three collaborative sites in our prototype system of collaborative solid modeling.

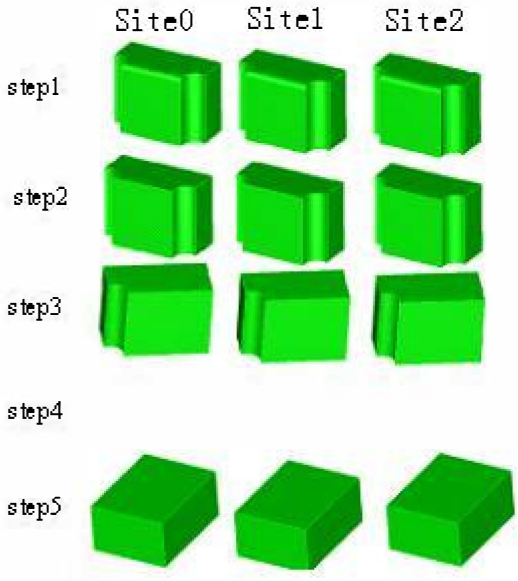


Figure 4. Multiuser Undo/Redo Process

Step 1: A base block is created by modeling operation Block(0) from site0. It's then sent to site1 and site2. Site1 sends operation Roundslot(1) to create a round slot on the left of the model then propagates it. Site2 creates operation roundslot(2) to create another round slot then propagates it to other sites. Finally site0 creates and sends command Edgeblending(0). The final boundary model is created after 3 sites' collaboration.

Step 2: Site0 sends an Undo command revoking the edgeblending operation it made. The command is carried out on local site and sent to site1 and site2 immediately. By properly locating the undo object on these two remote sites and model reconstruction, the model states on 3 sites are consistent.

Step 3: Site2 sends an Undo command revoking the round slot operation it made. The command is carried out on the local site and sent to the other 2 sites immediately.

Step 4: Site0 sends another Undo command to undo the base block creation operation it sent. It's quite obvious that round slot created by Site1 depends on this base block . Due to the elimination of this block, round slot is also eliminated.

Step 5: Site0 sends a redo command to redo the most recently command it has undone. Therefore, the base block is rebuilt by re-executing operation Block(0). Then the redo command is sent to site1 and site2. By properly locating the redo object on these two remote sites and model reconstruction, document states on these 3 sites are still in consistency.

VIII. Conclusions

This paper proposes a selective undo/redo mechanism in 3D collaborative modeling systems. To satisfy the fast response requirement, each site keeps a model state stream to keep the state at every step rather than the full ru-run mechanism. The mapping method of operations in history buffer and nodes in model state stream is proposed correspondingly. Finally, a set of undo/redo algorithms is proposed.

Acknowledgment

This paper is supported by the National Science Foundation of China (Grant No. 61070078) and the Fundamental Research Funds for the Central Universities

References

[1] Aaron Brown, "A recovery-oriented approach to dependable services repairing past errors with system wide Undo," PhD Thesis, University of California, Berkerly., 2003.

[2] Choudhary R, Dewan P, "A general multi-user undo/redo model," Proceedings of the 4th Conf. on E-CSCW, Kluwer:Academic, 1995, pp.231-246.

[3] Berlage T, "A selective undo mechanism for graphical user interfaces based on command objects,"in TOCHI, 3rd ed., vol.1, New York: ACM Press, 1994, pp.269-294.

[4] Sun CZ, "Undo any operation at any time in group editors," Proceedings of 2000 ACM Conf. on CSCW, New York: ACM Press, 2000, pp.191-200.

[5] Chengzheng Sun, "Undo as concurrent inverse in group editors," in TOCHI, 4th ed., vol.9, New York:ACM Press,2002, pp.309-361.

[6] Cheng Y. , He F.Z., Jing S.X., Huang Z.Y, "An Multiuser Undo/Redo Method for Replicated Collaborative Modeling Systems," Proceedings of the 13th Conf. on CSCWD, Washtington D.C: IEEE Computer Society, 2009, pp.185-190.

[7] Bin Shao, Du Li, Ning GU, "An Algorithm for Selective Undo of Any Operation in Collaborative Application,"Proceedings of GROUP'10, New York: ACM Press, 2010, pp.131-140.

[8] Bin Shao, Du Li, and Ning Gu, "A sequence transformation algorithm for supporting cooperative work on mobile devices," Proceedings of ACM CSCW 2010, New York: ACM Press, 2010, pp.159-168.

[9] Abowd GD, Dix AJ, "Giving Undo Attention," in Interacting with Computers, 3rd ed., vol.4, New York: Elsvier Science, 1992, pp.317-342.

[10] Prakash A, Knister MJ, "A framework for undoing actions in collaborative systems,"in TOCHI, 4th ed., vol.1, New York: ACM Press,1999, pp.131-139.

[11] Matthias Ressel and Rul Gunzenhäuser, " Reducing the problems of group undo,"Proceedings of GROUP'99 , New York: ACM Press, 1999, pp.131-139.

[12] Sun, C., Jia, X., Zhang, Y., Yang, Y., and Chen, D., " Achieving convergence, causality preservation, and Intention- preservation in real-time cooperative editing systems," in TOCHI, 1st ed., vol.5, New York: ACM Press, 1998, pp.63-108.

[13] Xueyi Wang, Jiajun Bu,Chun Chen, "Achieving undo in bitmap-based collaborative graphics editing systems," Proceedings of ACM CSCW 2002, New York: ACM Press, 2002, pp. 68-76.