SYNCHRONOUS 3D DOCUMENT COLLABORATION

A Thesis Presented to the Faculty of
the Department of Computer Science
Pace University
In Partial Fulfillment
of the Requirements for the Degree of
Master of Science

in
Computer Science

by

Lawrence Cayenne-McCall

November, 2008
Under the Direction of

Dr. Richard L. Kline
# Table of Contents

Table of Contents ................................................................................................................ ii  
List of Figures ....................................................................................................................... iii  
Abstract .............................................................................................................................. iv  
Introduction .......................................................................................................................... 1  
Historical Review .................................................................................................................. 5  
Transparent 3D Collaboration - Goals of the system ......................................................... 12  
The 3D Collaboration Approach .......................................................................................... 15  
Summary and Future Work ................................................................................................. 35  
References ............................................................................................................................ 37
List of Figures

Figure 1: MS Word 2003 tracked changes and inserted comments ......................... 6
Figure 2: Office Live Workspace ........................................................................... 7
Figure 3: Google Docs with two collaborators ..................................................... 9
Figure 4: SubEthaEdit with color coded users on Mac OS X ............................... 10
Figure 5: EtherPad in web browser and color coded users .................................. 10
Figure 6: SynchronousEdit in web browser with color coded users ..................... 10
Figure 7: Text Editor with tabs for layer access .................................................... 17
Figure 8: keyEvent Method .................................................................................. 21
Figure 9: mousePressed Method ......................................................................... 22
Figure 10: Alpha setting for sendBuffer ............................................................. 24
Figure 11: arrayBuffer ......................................................................................... 25
Figure 12: Top and Bottom Level Collision ....................................................... 26
Figure 13: Different Alpha Implementation Options ......................................... 26
Figure 14: Alpha blending issues ......................................................................... 27
Figure 15: Flag set for Conflicts .......................................................................... 28
Figure 16: Three users with different content at level 0 ....................................... 32
Figure 17: Repository with unresolved merges .................................................... 33
Abstract

This thesis on Synchronous 3D Collaboration describes an approach to managing documents with two or more collaborators. A group of collaborators need the ability to work simultaneously on documents. There are a number of ways the group can interact with a document. One may decide to divide work by person or small team. Multiple people may be designated authors while others can be reviewers and approvers. A work flow process may be created to manage the sequence in which documents and revisions migrate from one user to another. The group may work on many documents that make up a project. Finally, the documents the group creates may be part of a much larger group. The Synchronous 3D Collaboration thesis takes each of the ideas described above and allows collaborators to manage document editing with many contributors simultaneously.
Introduction

This thesis describes a Transparent Synchronous 3D approach to document collaboration. It builds on existing synchronous and asynchronous collaboration solutions. A synchronous solution is one that sends its information on a particular schedule; for on-line collaboration, this schedule may be real-time. The recipient can process information immediately and send information in the same manner. An asynchronous solution is one that does not have a schedule. The recipient does not know when the next communication will take place.

Existing synchronous collaboration solutions allow users work on documents at the same time. If one user makes a change in a document, others can also see the change. The Transparent Synchronous 3D approach also lets users collaborate on documents. One user can change a document that everyone is sharing, but these edits do not alter another’s document. Instead, each user gets a copy of all changes in a transparent 3D format and decides which content to accept into their own document. This approach merges synchronous and asynchronous communication into one solution.

The Transparent Synchronous 3D application is designed for a distributed team that wants as close to real-time synchronous communication while preserving asynchronous communication as needed. Its major change from existing solutions is its ability to allow a user to edit while viewing a transparent version of other collaborator’s work.

A team approach to document management has many advantages and disadvantages. A common way for a team to operate is to focus on the advantages of
collaboration such as breaking tasks into small components and assigning work to individuals. When the deliverables are disjointed and can be assigned distinctly to team members, team collaboration works quite well. Each person can complete assignments based on due dates and use necessary resources. As tasks become due, team members may spend time compiling the information into a single deliverable.

Unfortunately, team projects often do not get completed quite so nicely. Members may need to work on common documents at the same time, require feedback from others before continuing on their own task or work with many versions of a document. One who has worked on a team may be familiar with the frustrations of working under these types of conditions. Often, a group will try to coordinate activities to minimize collaboration [1].

There are three common ways teams worked together to produce deliverables. The first way is to work independently while periodically synchronizing results with others. This approach can be used when a team needs to review an entire document and provide comments. The second approach is to divide the work into disjointed sections and have the team combine results at scheduled intervals. Lastly, a workflow approach can be taken where one or more people complete tasks then pass the deliverable to the next person or team. This process can be repeated until all people or teams have their turn. These methods do not allow quick turnaround from team members and are most appropriate when the due dates are not very near [2]. These approaches are asynchronous.

Asynchronous data communication can be defined as data transmission without a schedule [3]. In each of the three methods described above, the users worked
independently without the use of an automated schedule. This asynchronous collaboration can be inefficient if quick turnaround is required. It is difficult for the group to maintain the status of the document as it makes its way from one person or group to another. Comments and content changes are done without the knowledge of others’ input in the group. This can lead to duplicates or conflicts. Finally, one may be ready to complete a task but may need to wait for other work to be completed.

The process gets a little better when an electronic document management system is used. A central shared repository can store document’s first draft. Security controls can be created to grant edit permissions. Logs can be maintained to let an administrator or document owner know who modified a document or currently has it opened. The biggest advantages of the migration to an electronic process are the ability to manage access to documents by more than one user and store multiple versions. After the group has gone through multiple iterations, the owner can review prior versions as needed.

Using electronic document management replaces an inefficient manual asynchronous process for one that can take advantage of automation. The design of automated systems uses the same manual concept along with some of the advantages of automation similar to the way the “desktop metaphor” is used to simulate the paper-based environment [4].

Teams have another approach to collaboration. They can use applications that let them work in a synchronous environment as if each were in the same room looking at the deliverable on a white board. This method allows for instant feedback and real-time collaboration but is not convenient for distributed teams.
The purpose of this paper is to describe one step toward a transparent 3D synchronous approach. The paper-based approach that it most closely resembles is a team watching an overhead projector presentation while providing content to one person. In this paper, we focus on synchronous communication that allows users to manage content while having visibility into others in near real-time.
Historical Review

The idea for a 3D synchronous document collaboration started with looking at existing collaboration solutions. These software solutions take two different approaches, asynchronous and synchronous. Asynchronous collaboration is the one most associated with most current word processing applications. Microsoft’s MS Word [5] is a typical single user application that allows collaboration.

Starting with MS Word 6.0 for Windows 95, Microsoft added asynchronous features that facilitated collaboration such as marking changes, reviewing comments inserted by others and managing document versions. The first of these allows each user to track changes and enter comments. As the document is circulated among other users who will use it in single-user mode, it tracks comments and edits made by color code. In this single-user mode, collaborators are responsible for coordinating with others in order to create a final document. If more than one version of the document gets created, MS Word can merge documents into one. The final document shows unresolved conflicts as tracked changes.

Figure 1 below shows MS Word 2003 with track changes and comments inserted. The text that has the standard black color was added by the document owner. Two other users have changes represented by orange and blue text. Any place where the original author’s text was removed is highlighted in the margin as a delete record. Comments have also been entered by users. Each user’s comments are assigned a sequence code along with initials.
MS Word also has two modes of collaboration which provide synchronous communication. First, in MS Word 2003 one can click *Tools* and *Online Collaboration* to launch a NetMeeting [6] session. NetMeeting is a third-party tool used for on-line collaborating. Microsoft incorporated this tool into MS Word 2003. During the NetMeeting session, control to a document can be passed to any user. Once a user takes control of the document, they are able to make edits while others can only view changes.

The second method of collaboration in MS Word is with a shared workspace option using MS Office Sharepoint Server [7]. This takes the single-user application and allows a user to share the document. As changes occur in any user’s version, Sharepoint periodically sends the updates to all versions. If a conflict is detected, users have the ability to view and accept them.
Microsoft has another product that enables document collaboration, MS Office Live Workspace beta. This is an on-line site that allows users share files and does not require MS Office on the local machine [8]. By setting permissions, users control access to documents. There is a single online document that all users share. All edits are done together as long as the user has permission. Conflict detection is not required because all eligible users share the same version. This is different than the Sharepoint solution where each user has their own copy and relies on receiving updates. Figure 2 shows an example of a typical session within Office Live Workspace.

**Figure 2: Office Live Workspace**

OpenOffice [9], a product sponsored primarily by Sun Microsystems, has two collaboration tools O3Spaces [10] and Paperiton DMS [11]. Another similar application is Documentum Collaboration server. All three are document management solutions that
rely on centralized storage, version control and file locking. This limits users to asynchronous communication.

There are several applications that have near real-time document collaboration without the need to share the desktop. The AbiCollab feature of AbiWord [12] allows users to edit the same document at the same time. AbiCollab works based on sending serialized Change Records (CR) to the other collaborators. It uses peer-to-peer synchronization so all collaborator’s changes are merged without the use of a central server. There are some issues raised by AbiCollab that had to be resolved such as, Internet lag, Undo, edits to others’ text and collision. These issues and their solutions will be discussed later in the document.

Mobwrite [13] and Google Docs [14] take a simple approach to collaboration and have an online word processor that allows one or more users to share the same document. With Mobwrite, synchronization is done every few seconds. Collision is handled by the application as best it can. It is understood that any mistakes will be visible and resolved by the users. Google Docs shows an icon of the other users currently editing the document. Users can be invited to collaborate by receiving an email from existing document users. Figure 3 shows an example of Google Docs with the primary document owner identified in the upper right hand. In the bottom right hand an additional collaborator is identified.
SubEthaEdit [15], EtherPad [16] and SynchroEdit [17] are tools that rely on one version of the document. Users who are currently editing the document are displayed. Figure 4 shows SubEthaEdit with its list of users currently editing the document on the right hand side and an edit control list on the left side. SubEthaEdit also has collaborative syntax for editing program languages. Kdawson, on the website Slashdot.org, says the major disadvantage of this tool is that it only works on Mac OS X [18]. Figure 5 shows EtherPad, launch in November 2008, with the same type of interface and functionality as SubEthaEdit but also works on Linux and Windows [19].
Figure 4: SubEthaEdit with color coded users on Mac OS X

Figure 5: EtherPad in web browser and color coded users

Figure 6: SyncrhoEdit in web browser with color coded users
The SynchroEdit application is a document editing system we have encountered that allows for synchronous communication and displays a way to track who is editing the document and what they changed.

The system proposed in this paper builds upon the best features of the SynchroEdit application and allows each user to see the changes made by another user as their full document. The main difference is that when collision is detected, changes are not made to another’s original document. Instead, altered text is displayed in the background as a transparent 3D image.
Transparent 3D Collaboration - Goals of the system

1.1. *Multiple collaborators should be able to edit a document at the same time.*

When there are multiple authors of a document, one author should not have to wait to access and edit the document. The system should allow multiple authors access based on permissions. Current systems work in one of two ways. Some environments allow one user to have edit permission while all other users have read access; this is done with file lock and release. Others allow all authors with edit permission access to the document at the same time. Each author can see the work of the others. The problem with this method is that there is only one version of the document at one time. If there is a conflict between authors, it has to be resolved immediately because all authors are sharing the same document.

1.2. *Each author should be able to edit a document without overwriting another author’s work.* One author should be able to work in a single-user mode while accessing changes made by another author in near real-time. The system should manage the interaction between an individual session and those of other users. This approach is similar to a group meeting in a conference room where each attendee is taking notes while an administrator is taking notes for the group discussion. Each participant would have a version of the document but would be sharing comments with the group.

1.3. *The users should be able to maximize content on the screen.* An author should be able to incorporate information from other participants. This is based on the idea that “software is too faithful to the paper-based world” [4]. In the paper-based model, a user tends to read a stack of paper one piece at a time; usually
starting with the piece on top. The user may decide to spread the paper across a
table to have access to multiple pages at a time. If more than one author of the
document needed to review one of those pages, they may take one of the
scattered documents to their part of the table. The original user who started with
the stack would not have access to the other documents. This is the paper-based
world.

On a standard computer screen, one has the ability to simulate the paper-
based environment with a typical single-user word processor. One can also
recreate the idea of spreading documents across a table by adding more displays
or by using the Windowing feature found in many operating systems. The
computer can do one thing that the paper based world can not do. Documents on
the computer can be manipulated in a way to allow transparent text so a user can
have multiple views on a single screen. They begin to take advantage of alpha
blending [20]. With alpha blending, multiple versions of documents can be
stacked beneath the owner’s original version. The partial transparency of alpha
blending allows the owner to see the other documents beneath their own. While
editing a page, one can still see the top level view. Transparent text from another
user may not be one hundred percent legible. If one wants to get details about
what is in this transparent text, it can be done so without disturbance.

1.4. Relaxed WYSIWIS capabilities [21]. In the relaxed What You See Is What I See,
there is not the server based approach that one gets with strict WYSIWIS
collaboration software systems. This means that multiple users are not sharing
the same document stored on a server where typically one user at a time is
granted permission to control the session. In the relaxed WYSIWIS model, updates are replicated to users. No one user has to be identified as the session owner. All users can make updates or review content to their own version of the document while getting updates from all other users.

1.5. The system should incorporate version control. As discussed by Stefanfia Leone [22], most word processors ignore document history. The collaboration system should allow users to switch among current versions controlled by any of multiple team members as well as their own history. In the paper-based world, it is very time consuming to review multiple versions of a document created by a single user, in order to consolidate the differences into a single document. This gets even more complex when the task involves multiple versions maintained by each individual within a team.

1.6. The system should merge content from multiple users. One of the most difficult tasks in collaboration is automating the merge process. The system should be able to take content from multiple users and show the differences in an efficient format or automatically create a merge document based on a set of rules and security settings.
The 3D Collaboration Approach

The 3D Collaboration approach is broken into five components.

I. GUI Presentation

II. Transparent Frames

III. Version Control

IV. Communication Manager

V. Security

I. GUI Presentation

Many word processors operate in a single user mode. The problem is that in today’s world, it is very common to collaborate with others to complete a document. With word processors, individual users have two basic options. First a user could lock the document, edit it then release the lock for the next user. This asynchronous method is usually accompanied by a synchronous process such as an instant messaging solution, telephone or live meeting. When a document is locked and others need to get access to it, a message can be sent to the current editor to inform the team when edits will be done.

In the approach for the 3D Collaboration, the GUI starts by looking like the typical word processor. When the document is first opened, a unique id is assigned. This id is used so that other collaborators can access the correct document. This is important when one or more users open multiple documents. Changes from one user need to get to the correct document. The next step is to assign metadata that will be used to track the different users, their access rights and location on the network.
Each document needs to have one primary owner. This is the person who initially created the document and its unique key. This owner grants the access rights to the rest of the team. Access rights can be granted to others that are identical to the original owner. This means that others may have full control even to grant access to additional users. In order to create the access rights, the owner must identify each team member by IP address or computer name. This is required to establish peer-to-peer communication.

Once the document exists, the unique id is created (system generated), and the metadata on users is added, the primary owner can begin creating the content for the document. As the user begins to type, the first thing they will notice, which is different than most word processors, is a frame of icons listing others users who are currently accessing the document. Each user’s icon also has a username assigned. This is the same name the primary user initially used when assigning access rights. The user frame will be discussed in more detail later in this section.

The GUI Presentation is responsible for displaying the content for the primary user as well as the content for all users who have permission to edit the document. As stated earlier, one approach to collaboration is the strict WYSIWIS based on the chalkboard. This approach lets users know who is logged on and what they are doing. But, it has drawbacks such as collision, undo and merge because there is only one version of the shared document. The GUI in the 3D application doesn’t face the same problems as other synchronous applications. This is because it allows each user to have a version of the document. The GUI displays each version of a document as a transparent layer. The top layer is each user’s own version. The layers below the top layer belong to other users.
The top layer user can switch layers by selecting the tabs at the bottom of the screen as displayed in figure 7 below. This is important for two reasons. First, when there is a collision situation, that is, two or more users have different content that occupies the same space, the layer that is immediately behind the top one is displayed with a semi-transparent setting of alpha 0.2. The top layer always gets a value of 1.0, which is no transparency. With a setting of 0.2, the second layer can be read by the user currently working on the top layer. For typical black and white text, a setting of 1.0 looks like ordinary text and 0.2 has the effect of grayed-out text. The setting of 0.2 can be assigned to any level within the layers of the GUI.

Figure 7: Text Editor with tabs for layer access

A value of 0.2 can only be assigned where there is collision. This is because of the negative effect of alpha blending in which the content on top becomes distorted. Microsoft Research looked into this problem as they tried different methods for
maximizing content on a single display [20]. They found that the top level content can become difficult to read when additional layers are also presented. Their solution differs from the one in this paper because they primarily were working with text on top of graphics. Although their solution will differ, the original problem still persists. Therefore, the solution used in the 3D application is to limit alpha values to 1.0 when the top version occupies a space by itself. The 0.2 setting is used to show an additional layer at any level as long as no other level is also occupying that space.

The objective of the GUI is to give the user synchronous access to all available content. While the top layer is being accessed by the top layer’s owner, other layers are also updating content through a separate thread every 500 milliseconds. If the top layer user wants to see detail content from another layer, they can click on that layer to bring it to the top. The active layer then becomes the second layer.

At the GUI Presentation level, automatic merges are not done. All collision resolution and white space is handled by the user in their own top layer. If one user makes a change to a universally accepted section, their edits are made in their level 0 and all other users will see the edit in level 1. For example, user A will not see the content from user B if collision is not detected. When collision is not detected, both users have the same text and both see their own content as the top level. When collisions do occur, users A and B have different text in the same place on the document. Either user A or B can accept the other user’s text or change their own text to match the other user’s text. This resolves the collision. Once the content is accepted, it is changed to level 1.0. Once content is set to level 1.0 for all users, where all users have accepted the content, then all users will share the same baseline version for that section.
Currently, it is up to each user to resolve collisions. This is similar to asynchronous methods using current word processors and a document management system. In these environments, a version of a document is checked-out by one person. The user views comments by others and makes edits as needed. This process may be done by an individual or a group in a conference room sharing a projector. An asynchronous document management system allows for collision resolution in a slow manner. A process such as a group gathered together in a room sharing a monitor or projection display allows for collision resolution but requires everyone to be together. The solution proposed here is to allow fast synchronous feedback for a dispersed group while still relying on individual users to resolve the collisions as they occur.

There are methods available for collision detection and resolution. They are mainly used for two types of systems, strict WYSIWIS and version control. In strict WYSIWIS systems, everyone is sharing the same version of the document. For example, in Google Docs, there is only one version of a document. All edits are performed on the same version. If two or more users want to make edits, they may collide if they try to access the same content. Due to internet lag, one user may try to access content that another user is trying to access without knowing what each other is doing. A user may try to change content that another user has deleted. Some unpredictable behavior may appear to occur when this happens. AbiWord resolves this by using sequence identifiers [12].

Another environment that tries to resolve collision is within version control such as Subversion (SVN) [23]. This system uses the copy-modify-merge method. One user checks out a copy and adds their content then submits their final file. At the same time,
another user can copy the same file and make edits. When they submit their version, SVN will attempt to merge the edits. In cases when it can’t resolve differences, it stacks the content and inserts comments about the collision.

The primary goal of the GUI is to display content for a single user as well as the content of other users. This is done to maximize the content available on a single screen and replicate what multiple users would do in a paper based world if they could actively see what everyone else is working on. The GUI layer for one user does not receive data directly from another user’s GUI layer to render the 3 dimensional displays. Each GUI layer receives its information from the transparent layer.

II. Transparent Frames

The transparent layer is responsible for the 3D graphics displayed on the screen. The work of this layer is divided into four sections. First is the text panel that is displayed on the screen. Second are the events such as keyboard and mouse. The next part is the type of graphic and its location. The last section decides what type of font to display.

The Java JPanel class is used which is a generic lightweight container. This gives the user the basic virtual sheet of paper to type on. It is on this panel that the entire GUI gets displayed. Although the GUI is described as a separate layer, it is really part of the panel. The two different layers are separated into what you see on the screen, the presentation, and what the system is doing in the background to generate that display.
The panel implements the KeyListener() and MouseListener() classes. The KeyListener() is used to get the events from the keyboard. As a user types, each event is captured by the keyTyped method. This method gets one character at a time and is responsible for passing it to the graphics part of the panel for display. It is only responsible for capturing events from the top level. It does not know about key events that occur on other collaborator machines. Once the method has the character it passes it to a buffer where it waits for its turn to be written to the screen. All Unicode characters are accepted. **Enter, Backspace, Delete** and **Space** keys have their standard meaning as they do in other word processors.

A buffer is used to store each key typed by the user on the local machine. Each character in the buffer is written separately. A word in the buffer is defined when the user types the **Space** key. A new row is defined when the user types the **Enter** key. When this occurs, a new buffer is added. In the current implementation, the buffer is stored in an ArrayList where each element represents a row on the screen. Figure 8 below shows the KeyEvent method and the ArrayList sendBuffer object.

**Figure 8: keyEvent Method**

```java
public void keyTyped(KeyEvent e) {
    ch = e.getKeyChar();
    switch (ch) {
        case '': {//delete one character from current line
            text.deleteCharAt(text.length() -1);
            if(text.length() == 0){
                sendBuffer.remove(idx);
                idx = idx -1;
                text.append(sendBuffer.get(idx));
                sendBuffer.set(idx, text);
            }
            break;
        }
        case '
': {//add new row
            text = new StringBuilder(" ");
            idx = idx + 1;
            sendBuffer.add(idx, text);
            break;
        }
        default: {//keep typing on current row
            text.append(ch);
        }
    }
}
```
try {
    sendBuffer.set(idx, text);
}
catch(IndexOutOfBoundsException c) {
    sendBuffer.add(idx, text);
}
break; }
//end switch
this.repaint();
leftMargin.repaint();

Because each character is written to the buffer separately, the entire string that makes up a line is treated as one long text. For now, this is required so the user can see each character on the screen as it is typed.

The next event type that is used by the panel are those that control down and up clicks and dragging from one location to another. When a user clicks the mouse, the current (x, y) coordinates are saved along with the information stored at that location. Currently the information is an entire line. As noted above, that will be at the word level. As the user moves the mouse across the screen, the (x, y) coordinates are tracked and passed along with the buffer to the object responsible for drawing to the screen. This allows the user to move content using the mouse and dragging it where ever it needs to go. Figure 9 shows the method for the mouse event handling.

Figure 9: mousePressed Method

public void mousePressed(MouseEvent e) {
    saySomething("Mouse pressed; # of clicks: ",
    + e.getClickCount(), e);
}
public void mouseReleased(MouseEvent e) {
    saySomething("Mouse released; # of clicks: ",
    + e.getClickCount(), e);
}
public void mouseEntered(MouseEvent e) {
    saySomething("Mouse entered", e);
}

The keyboard and mouse events give the user the ability to navigate through the top level as well as transparent levels. By clicking on words within the transparent levels, the user
can temporarily switch the view to other levels or accept other levels into their own level permanently. They can use this method to resolve data conflicts or review another’s content. By clicking on the names or icons of other collaborators in the tray, the content of entire levels can be brought to the top layer.

The next part of the transparent layer is the graphic that is displayed. At the time the user types a key, the font, size and type are not stored in the buffer. The buffer only stores Unicode characters. The paintComponent() is responsible for setting the font, size and type to write to the screen. This information can be defined by the user. At this time, it is defined by the level; all content within a level must share the same format. The paintComponent() also sets the alpha value and coordinates.

The buffer is accessible by the paintComponent and ultimately gets drawn by this method. Every character that is added to the buffer goes through the paintComponent at the time it is typed. The component determines if the buffer has changed and if a redraw to the screen is necessary. Previous buffers that are written to the screen, which have no change, are not redrawn. Other buffers that are actively receiving Unicode data from the keyboard, get redrawn each time a character is added.

The paintComponent defines the alpha setting. This value is defined from 0.0 to 1.0 and determines if the graphic is transparent, opaque or somewhere in between. Throughout a user’s session, the alpha values change depending on where the mouse and keyboard are clicked. When the user is working in their top level, the alpha setting will be at 1.0. When another user begins typing, their top level will also be set to 1.0 but will be passed on to other users as 0.2.
Figure 10: Alpha setting for sendBuffer

```java
if (!(sendBuffer.isEmpty())){
    for(int i = 0; i <= idx; i++){
        if (row == i || sw != 0){
            g1.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 0.2f));
            Font f2 = new Font("Serif", Font.PLAIN, 30);
            g1.setFont(f2);
            layout = new TextLayout(sendBuffer.get(i).toString(), f2, frc);
            layout.draw(g1, 10, line);
        }
        else{
            g1.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 1.0f));
            Font f2 = new Font("Serif", Font.PLAIN, 30);
            g1.setFont(f2);
            layout = new TextLayout(sendBuffer.get(i).toString(), f2, frc);
            layout.draw(g1, 10, line);
        }
    }
}
```

Figure 11 shows the two methods that define levels 0 and 1. Level 0, defined for the user’s current view, gets its content from the sendBuffer and writes the output using layout() method at the coordinates specified by the X, Y parameters. These parameters are passed to layout() from the X, Y coordinates captured by the keyboard events. The X, Y values are not stored after the layout() completes the rendering on the screen. They are used once and discarded. The buffer and the array take care of knowing where the characters are on the screen without the need to store the X, Y values. The X value is currently set to 10 which allow each row to start writing in the same place. The buffer takes care of moving the X value as characters are written to the screen.
The buffer is one string per row in the word processor. Each string can be accessed as an entire row. Or, the string can be accessed one word at a time using substrings and identifying blank spaces. This is how the Y value is calculated.

Mouse events will also work within the buffer and array to allow movement of word or sentences. Each mouse event has X, Y values assigned as the mouse moves across the screen. When the mouse stops on a word, the X, Y coordinates of the mouse are known. But, the X, Y values will not line up with the buffer and array. This is because each character is made up of pixels, each consisting of different coordinates. Therefore, when a user clicks on a character, there isn’t one spot that identifies that character’s coordinates. To get around this problem, the character’s metrics will be used. This defines the space required to write the character relative to other characters. If the user clicks anywhere within this space, the mouse click is considered successful and compares its X, Y to the nearest X, Y in the buffer and array.

The last part of the transparent layer is the collision detection. This takes the buffer from two or more users at the same row and decides if there is a collision or white
space. Currently it does this for the entire row. The figures below show the different approaches that can be used. The first one is selected for this implementation.

The example below, in figure 12, uses the current implementation shows that collision at level 0 pushes other levels into transparency. This is done in a way to allow the user at level 0 to see other collaborator’s content while still maintaining the ability to read their own. Even where there is collision and no white spaces, it is still possible to make out the words.

**Figure 12: Top and Bottom Level Collision**

This is the top row that has different content.

The example below shows what happens when the user clicks on the level one text. It switches the transparency from level 1 to level 0.

**Figure 13: Different Alpha Implementation Options**

*Current implementation: collision with one other user (level 1)*

This is another row that has different content.

No collision is detected in the second row of level 0.

Collision is detected in the third row of level 0 but in a white space. Level 1 is longer than level 0.

*Another option A not used.*

This content only goes to the end of this word.

*Another option B not used.*

This content only goes to the end of this word.

*Collision with multiple levels*

This content only goes to the end of this word.
Another noticeable feature is that the font size of level 1 and 2 is smaller than level 0.

The characters are also shifted up and to the right. Both of these effects are done to allow the illusion of dimensions as well as make the text legible. The examples below show the effect.

Figure 14: Alpha blending issues

This is level 0 but it is difficult to read level 1 below.

This is level 0 but it is difficult to read level 1 below.

The last scenario may not seem obvious but is very important. When multiple users open a document for the first time, they are all accessing the same version. Therefore, all current content in the document would normally be viewed as a collision and trigger transparent layers. To avoid this situation, each line gets assigned a flag which indicates if it has been updated from the version retrieved from the version control. The default setting is No where the line has not been altered. When one user makes an update to a line, the flag is changed to Yes. A Yes flag triggers transparent versioning within the panel. Below is an example where users A and B kept the original line and C made a change. A and B would see a transparent level for C but not each other’s level.
The goal of the transparent layer within the Java panel is to create an environment that gets away from the paper-based world while giving the collaborators enough of each others’ content to make it useful. The next module in the 3D collaboration looks at the effect of document management.

III. Version Control

Most current text editors are considered single-user applications. In this mode, document management is as simple as create the content and save the document. When collaboration is included in the environment, a single-user turn into multi-user where version control is required to manage each user’s content. Without version control, version collision would be common and collaboration would fail. Take an example with two users, A and B. The two users can access the same content in one document on a network file server. User A can edit the document and save the changes. If user B also has a copy of the same file and makes edits, there is no way to know what edits user A made. Therefore, user B’s changes will be the only ones that remain in the file after both users save their content.
There are two ways version control handles this situation for document management. The first method is Copy, Edit and Lock. The second is Copy, Edit and Merge. The first method is quite common within document management systems but requires collaborative work in a serial manner. The first user to open a document locks it while edits are occurring. Once the edits are complete, the user can replace the repository version with the new one they just created. This releases the lock for other users’ access. If two or more users need to edit the same document, they must wait until all locks are released. This is also the case when multiple users want access to the same document but may want to update different sections.

The second and most effective method allows two users to access a document at the same time. This method uses Copy, Edit and Merge. All users have access to documents as needed and update content. The difference is when users are ready to save. As versions from each user’s copy are updated within the repository, the version control application looks at differences. When it detects white space, it merges the content. When collision is detected, it does its best job to insert all content without losing anything. At this point it is up to a document owner or system administrator to look at the final copy in the repository and manually correct any unresolved merges.

Although this is the preferred method for version control, there are three drawbacks. The first is automatic merges by the system is risky. Merging content from a draft essay file may not be too difficult. Merging content from a computer program such as Java is a little more difficult. This is because many times, each character in a program is very important. The sequence of characters, content and punctuation are critical when it comes to compilation. Sometimes a merge can take place that will insert data in the
wrong sequence without triggering an error during compile-time. But, the actual results of the program during execution may be undesirable. Finding and correcting these errors can be challenging.

Second, depending on the type of document and its content, the merge can lose data. A user may need to review a previous version of the file to determine what changed. Then, they can manually compare the current version with the archived one to correct the mistake of the merge.

Third, and probably most significant in terms of deficiencies, is that multiple users can access a document at the same time and know who the other current users are. They can allow the system to assist in the merge when each user submits their copy back to the repository. But, during the time that each user is changing the content of their copy, no one else has visibility into what others are doing. This is critical when a user’s content is dependent upon edits others are making. As mentioned earlier, this is an asynchronous method of document management. This works well when the due date is not immediate. This allows additional time to make edits, submit to the repository then check out the copy again at a later time. This process is repeated until all users have confirmed their edits.

The synchronous method proposed by the 3D solution solves this problem while still taking advantage of version control using Copy, Edit and Merge. Each level gets tracked by version control as a separate document. The user operating at level 0 sees their own document at the top while all other documents are either semitransparent or available on the tabs. All levels are tracked by version control but each user is only responsible for updating the repository with their top level version.
As each user edits their own top level, they have visibility to other versions through the levels. They use these semitransparent levels to manually merge collisions. As entire rows get resolved to one version, the levels in that row are removed. If all collisions have been removed prior to the final commit in the repository, then one file is updated and no further work is needed.

When all content does not get resolved prior to the commit within the repository, then each user’s top level gets submitted using the copy, edit and merge technique of the repository. At this time, a user may experience an automatic update or may get prompted to make a decision prior to the commit. These options are dependent upon the version control implemented with the 3D application.

Figure 16 below shows three users, Pink, Red and Yellow. Each checks out a copy from the repository and has their level 0. During the course of the session, the versions become different. Each user sees the levels from their collaborators (each level is color coded to show the appropriate view) and can actively resolve conflicts. By the time we get to the next section, all versions are the same and each user only sees their level 0. All three versions are submitted to the repository where the copy, edit and merge process ends with one file in the repository.
The next example, figure 17, shows the same three users. In this case, Pink resolves differences with Red in its level 0. Red gets its Pink level 2 automatically removed but Yellow remains. Yellow gets its Red level 2 automatically removed. The only conflict that remains is Yellows conflict with Pink, which can not be resolved during this collaboration session. The Yellow/Pink conflict also means that Red has a conflict with Yellow too. The result is that the merge in the repository consolidates Pink and Red and does not know what to do with Yellow. This must be manually merged when the users log into the 3D application again. Every user, except Yellow, will see a version of Yellow’s content in their levels.
IV. Communication Manager

The next component of the 3D application is the network communication manager. This is used to allow synchronous and asynchronous communication among collaborators. The synchronous communication is accomplished by allowing users to transmit and receive content as it is created. The asynchronous communication is used by the repository allowing users to transmit updates which others will access at other times.
To transmit content as it gets created, the users must belong to a network either through a LAN or VPN. The communication is handled by Java sockets. A socket is an endpoint of a two way communication on a network (a server and client communication). In the 3D application, this means that one session will function as the server while one or more user sessions will function as the clients. The server will typically be on the same server as the repository and should always be running. It’s constantly waiting for a message from a client and will distribute any message out to all others connected at that time.

The server side runs on port 6013 and listens on that port for clients who connect using a known IP address. The server is multithreaded so each client that connects gets its own session. This allows the server to receive content from one user across a communication channel and transmit this across to others.

V. Security

The last part of the package is the security group. This allows the owner of the content to assign other team members with access rights to the documents. By default, each user invited to participate in the group has full access. Unlike other synchronous collaboration tools that allow all users to share one document using WYSIWIS, the 3D application gives each user their own copy with read-only access to other user’s content. They are responsible for editing their own content but never change content at other levels. It is only within the version control repository where each user’s content may get altered.
Summary and Future Work

This research focused on 3D collaboration within a simple text editor. Future work in this area would continue in three main directions. The first is expanding the scope to include other word processors. The next step would be to focus on additional functionality to enhance collaboration. The final area would move this approach to other tools such as presentation tools, spreadsheets and other single-user applications.

The work completed in this thesis should be applied within a full featured word processor. The word processor selected as a next step would have an impact on the programming language selected. For example, if MS Word was used, then C# may be the language of choice. Once additional word processors are integrated, another implementation should allow collaborators to use the word processor of their choice. For example, a collaborator using MS Word should be able to work with a teammate using OpenOffice Write.

Additional functionality should also continue to be researched. The current scenarios described in this paper focus on two users, limited content collision and one document. Other scenarios should be included such as a collaborator who is working on multiple documents at the same time. Each document could have different users assigned to them. A large team may also be involved in modifying a document. When large teams do exist, this also may cause significant edits.

It is very likely that a team working on a particular deliverable may need to collaborate using more than just a word processor. They may also need to work on spreadsheets and presentations. And although there are still many challenges with Alpha blending with video, a final next step would be to apply this to video displays.
Each of these next steps, along with the current research, attempt to take solutions that exist today and move them closer toward a single environment. This would be equivalent to having an online meeting where everyone’s desktop is being shared. Included in this would be an expanded desktop taking advantage of 3D displays. Along with all of this would be another step which takes the model just described and allows collaborators to pass a document to any other team member. This is like sitting at a desk and passing a hard copy document to a co-worker to review.

The goal of this project has been achieved; to build a prototype capable of allowing two collaborators to work on unique documents while viewing the content of the other person. One collaborator can view another’s text as semi-transparent, slightly off-center and smaller font. While doing this, the collaborator can maintain their document’s content or chose to accept changes made by others.
References


