Collaboration Software

Steven Colgrove, Adam Svoboda: K-State Salina

The Problem

Currently there are several different collaboration tools available online. These tools can be useful when working on group projects, but each come with its own unique set of strengths and weaknesses. Generally, tools such as TeamViewer and Remote Desktop do a good job of sharing a screen, but force the users to share a single mouse cursor, which can be frustrating. Additionally, they require the user to share an entire computer desktop instead of a single window, which could be a major security issue.

The Goal

The goal of this project was to create a piece of software that would allows users to easily and securely collaborate on group projects without having to give up control over their mouse cursor or entire desktop. The following list shows the basic requirements for this project:

- Transmit encrypted screen data from one user to another
- Transmit encrypted input data from one user to another
- Traverse NAT to allow connections behind strict firewalls
- Process all transmitted data in real-time
- Provide a way for a user to connect and start a share with another user
- Show individualized mouse cursors for each connected user
- Run on all Windows operating systems XP and later

Methodology

There were several methodologies used in the creation of the project. Prototyping was used heavily in the early stage of the project to help hash out the most effective method to accomplish some of the more difficult tasks. Due to the complexity of the project and the likelihood of frequent changes to be performed, a chief-programmer style of programming was implemented in the early stages of the final product. Once the overall project reached a more stable state, pair programming was used. This allowed for the more difficult problems to be solved using the different perspectives and strengths of each of the programmers. Each methodology used provided unique advantages that were beneficial to the completion of the project.

The Solution

The final project resulted in a desktop/window-sharing tool backed by an asynchronous client/server system. Main features include:

- The possibility to share the entire desktop or a specific window with one or more individuals in real-time.
- Individualized, custom mouse cursors, isolated at a window-level to prevent interference with background tasks.

Technical Overview

All desktop/window capturing functionality was accomplished through heavy usage of the Windows API. In order to utilize the Windows API from the .NET language, platform invoking is required [2]. Frequent calls utilized in screen capture are outlined below.

GetWindowRect	Used to get the boundaries of a window object in order to prepare for the screen capture.
GetWindowDC, CreateCompatibleDC	Acquire the device context of the window and create a compatible device context for the bitmap.
BitBlt [1]	Perform a bit-block color transfer of the data from the source device context into our bitmap object
PrintWindow	Used to send a redraw to windows whose borders are not rendered using GDI, but DirectX (Windows Aero/DWM, Vista+).

Technical Challenges

One of the most challenging parts of the project was a result of a TCP socket size limitation in Windows XP. Initially, images transferred over the network were only partially received on the other end. The maximum TCP buffer size in Windows XP is 17,520 bytes, and images exceeding that amount. To fix this issue, a check was implemented to split the image into smaller pieces before sending the data over the wire. In addition, TCP message framing was added so that the client-server model understood the packet boundaries. Data was sent after being serialized with protobuf, Google's open-source data exchange format [3]. Message framing was implemented by sending the full packet size in the beginning of each packet header.

The multiple mouse cursors required a way to draw a customizable mouse cursor on top of anything, and limit its boundaries to a specific window. In order to accomplish this, a "mouse canvas" was drawn on top of the target window, which was a borderless, transparent window in itself. On top of this canvas, anything could be drawn. This was used to host the custom mouse cursor objects and keep them confined within the windows boundaries.

When transmitting image data, maintaining low bandwidth consistency and reducing line congestion is very important. The software was designed to automatically discard repetitive data and only send data that has changed since the last screen capture. This keeps line congestion low and reduces overall bandwidth consumption. If the computer's screen is idle, no data will be sent over the wire. With these optimizations, the software was able to transmit compressed youtube video at 1080p at a consistent rate of 60FPS over a LAN connection.

Bibliography

1. http://msdn.microsoft.com/en-us/library/windows/desktop/dd183385%28v=vs.85%29.aspx, "Bitmap Functions", MSDN
2. http://www.pinvoke.net/
3. http://code.google.com/p/protobuf/, "Protobuf – Google's data exchange format"

Collaboration Software

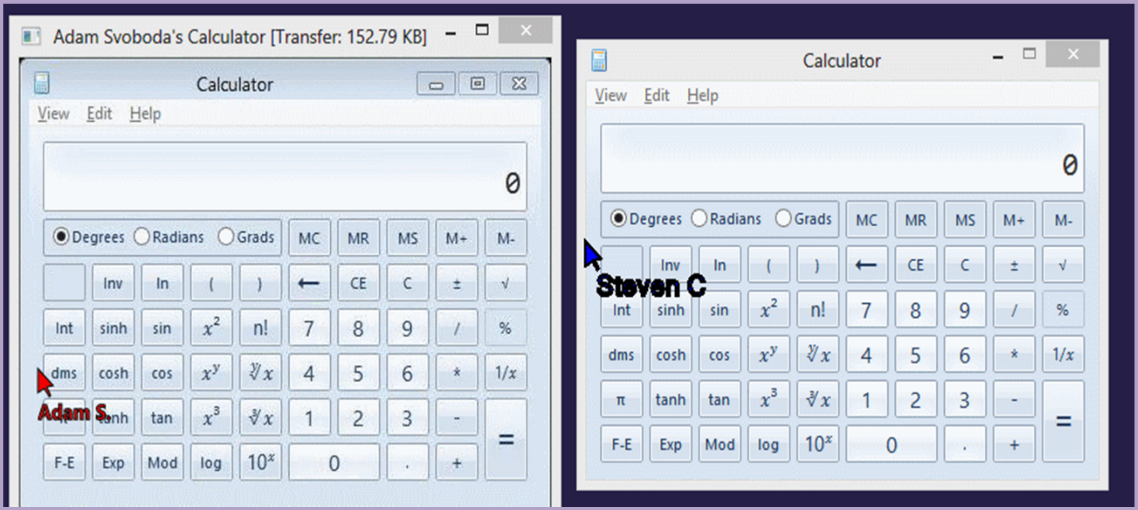
K-State Salina, CMST Senior Project
Steven Colgrove & Adam Svoboda

The Problem

Software such as Remote Desktop and TeamViewer do exceptional jobs at sharing the entire desktop, however they aren't able to restrict themselves to a specific window.

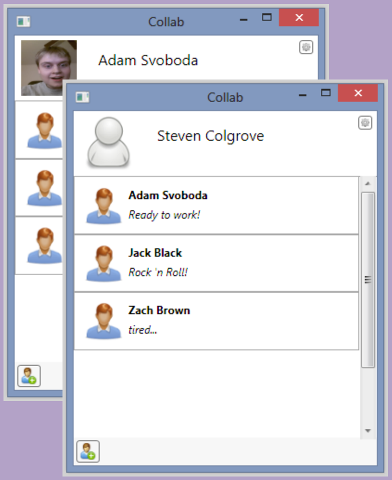
When using screen sharing software, multiple users fight for control over the same mouse pointer when using these products in a group environment.

Collaboration with two or more individuals can become difficult and intrusive.



The Solution

- Possible to share the entire desktop or a specific window with one or more individuals in real-time.
- Each user acquires their own mouse cursor to relieve the awkwardness of potential input collision that can occur when multiple people try to use the same device.
- Once connected, users can control the same application seamlessly, with individualized input.
- Shared cursors or applications are isolated at the window-level, to prevent interference with background tasks.



Behind the Scenes

- We only transmit data in the window that has changed since the last transmission. This saves bandwidth and improves overall performance.
- Multiple mouse cursors are drawn on a hidden, transparent window that sticks on top of the window being shared, which allows click pass-through.
- Bitmap data is compressed in the JPEG format.
- Screen/window data is broken up into smaller bitmaps before its sent across the network, to avoid problems that result from transmitting large buffers on some machines.
- Asynchronous client/server model with TCP message framing.