IMPLEMENTING GAZE-CORRECTED VIDEOCONFERENCING

Jim Gemmell
Microsoft Research
455 Market St.
San Francisco, CA, 94105, USA
jgemmell@microsoft.com

Dapeng Zhu
Stanford University
Computer Science Department
353 Serra Mall
Stanford, CA, 94305-9025, USA
dapengz@stanford.edu

Abstract

Lack of gaze awareness is a key failure that hinders the widespread acceptance of videoconferencing. GazeMaster is a project which attempts to provide a software solution to gaze awareness and eye contact. Previous publications have described the general approach of GazeMaster. This paper describes our implementation experience, giving more details of our software architecture, and explaining our approach to head modeling. Our results with respect to video, graphics, and networking are solid. Our computer vision technology, while promising, is still immature, and requires further research.

Key Words

Gaze, video-conference, computer vision, 3D modeling

Introduction

GazeMaster is a Videoconferencing system designed to provide eye-contact and gaze awareness. An in-depth discussion of its motivation and design is presented elsewhere [1]. In this paper, we describe our experience implementing GazeMaster. Software issues are covered in detail, and remaining open issues are enumerated.

Gaze awareness and eye contact are important to face-to-face communication, providing signals for turn taking, and also for creating a favorable impression[1] [2, 3]. Unfortunately, most videoconferencing systems do not provide gaze awareness or eye contact. This is due to the fact that when the user is looking at another party on their screen, they are not looking in to the camera (Figure 1, top). The image of someone looking away from the camera will never appear to make eye contact (Figure 1, middle), while the image of someone looking into the camera will always appear to make eye contact (Figure 1, bottom).

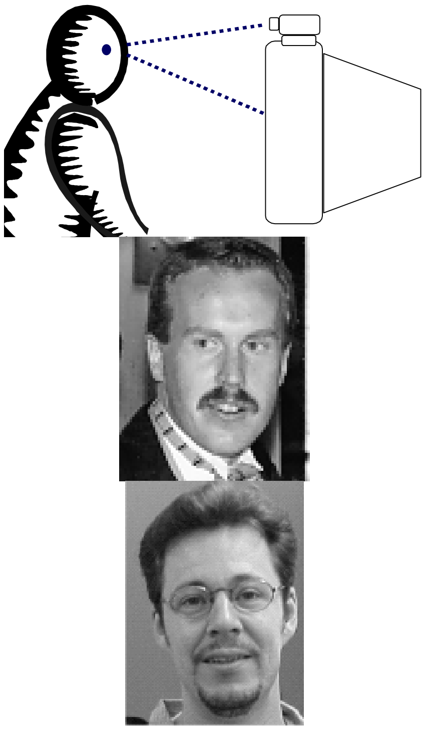


Figure 1: Top: gaze directed at display, not at camera; middle: not looking at camera – no eye-contact; bottom: looking at the camera – eye contact

Another problem that videoconferencing faces is lack of ubiquity: if no-one else has a videoconferencing system compatible with yours, it is not much use. Therefore, a critical goal for GazeMaster is that it be feasible as a software upgrade to common hardware: a PC with a single camera. This precludes approaches such as morphing from stereo views [4] or using special hardware (e.g., half-silvered mirrors – see [1] for a description of hardware alternatives).

[1] Adjectives for people using increased eye contact include confident, mature, and sincere, while those using less eye contact are described as cold, pessimistic, defensive, immature, and evasive.

GazeMaster's innovation lies in the processing of each video stream. In each frame of video, computer vision techniques are applied to detect the position/orientation of the head and the outline of the eyes (this is referred to as "segmenting" the eyes). This computer vision data is sent across the network along with the video frames. On the receiving end, the computer vision data can be used to alter the video image: eyes can be cut out and replaced with synthetic eyes, and the video of the face can be texture-mapped on a head model, allowing the head to be rotated in virtual 3D space. Figure 2 illustrates GazeMaster video processing.

Our head-pose tracking system is based on [5]. We have attempted to further the state of the art, for example in eye segmentation [6], and in feature detection [7]. However, computer vision remains an outstanding problem for GazeMaster. None of the techniques we have used are sufficiently robust for a wide population of users in everyday environments. GazeMaster awaits further maturity in the computer vision field before it can be widely deployed.

In the remainder of this paper, we describe our implementation of GazeMaster in more detail. We cover the software architecture and APIs used, the approach taken for head modeling, and special features required to debug such a system.

GazeMaster Software

GazeMaster leverages a number of application programming interfaces (APIs) in the Windows Platform SDK. DirectX SDK is used for 3D rendering. DirectShow is used for media flow, and to take advantage of standard video codecs. WinSock is used for networking.[2]

The current version of GazeMaster uses IP multicast for network transmission. For simplicity, packets are simply encapsulations of DirectShow media buffers. In our LAN setting, loss was not an issue. To make the software more generally applicable, it would be advantageous to use a standard for video transport such as RTP (this would allow non-GazeMaster-enhanced clients to at least render the raw video). Loss should also be addressed. GazeMaster supports H.263, H.261 or motion-JPEG video coding, and any other DirectShow codec could be easily substituted.

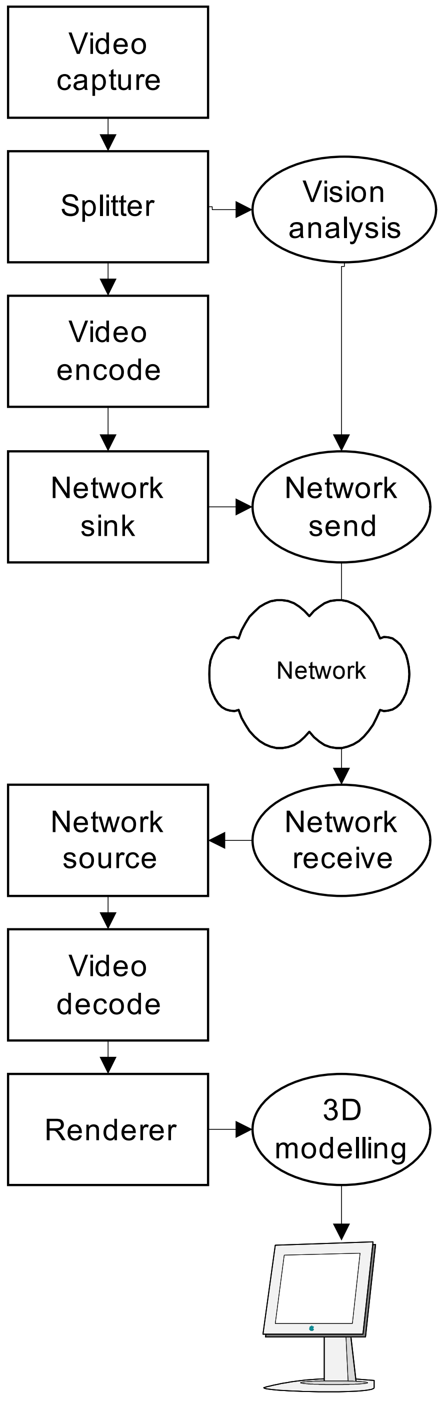


Figure 2: Video processing in GazeMaster. Rectangles denote DirectShow filters; ovals denote c++ classes.

DirectX takes advantage of the 3D acceleration in modern display cards, making the CPU burden for 3D rendering negligible (on a 1GHz Pentium, rendering added only 2-3% processor load). Each video frame must be decoded, and copied to a buffer for use as a texture-map. Points in the texture map are indicated as corresponding to each vertex in the head model. After this, the 3D hardware does the rest.

[2] Information on these SDKs can be found at msdn.microsoft.com

DirectShow is an API for media playback, transformation, and capture. At the heart of the DirectShow is a modular system of pluggable components called filters, arranged in a configuration called a filter graph. A component called the filter graph manager oversees the connection of these filters and controls the stream's data flow. Each filter is a COM component, with media IO interfaces called “pins” (after the pins on an integrated circuit chip). For details about DirectShow, the reader is referred to the Microsoft Platform SDK documentation. For the purposes of the paper, it is sufficient to understand that GazeMaster uses some standard DirectShow filters, creates some new ones of its own, and controls data flow via DirectShow.

Each GazeMaster client runs two pieces of software: and outgoing component and an incoming component. The outgoing component captures audio and video, runs the vision component on the video to determine head pose, eye segmentation, and eye gaze direction, and sends the audio, video, and vision data across the network. It also captures and transmits audio. The incoming component receives audio, video and vision data from other users across the network. It uses the vision data along with the video to modify the head and eyes in each video frame so as to give the desired head pose and eye gaze. It also receives audio from the network, and plays it using Microsoft DirectSound to locate it in 3D space. Figure 2 shows GazeMaster’s video architecture.

The GazeMaster Head model

The 3D head model used by GazeMaster is intentionally simplified. To texture map video images onto an extremely accurate head model would require robust tracking of many features of the head. For example, suppose we modeled the cheek with its muscles. To accurately map the video of the cheek onto the 3D model, we would need to track enough points on the cheek to manipulate the cheek muscles in the model. As we will discuss later, it is extremely challenging to track even the easiest features (the cheek, for example, is very difficult to track due to its uniformity in color). To avoid this stringent vision requirement, we use a simple head model.

The simplest head model would be a plane. However, when rotating a plane in 3D space, no occlusion will occur as the head turns, preventing the perception of a turning head (instead, the user perceives it much as viewing a portrait from the side – the head pose is interpreted as being the same as in the straight on view). As the head turns, it is important that occlusion occurs due to the curvature of the head, and prominent features of the head like the nose. That is, as the head turns away, the far cheek and far side of the nose should not be visible.

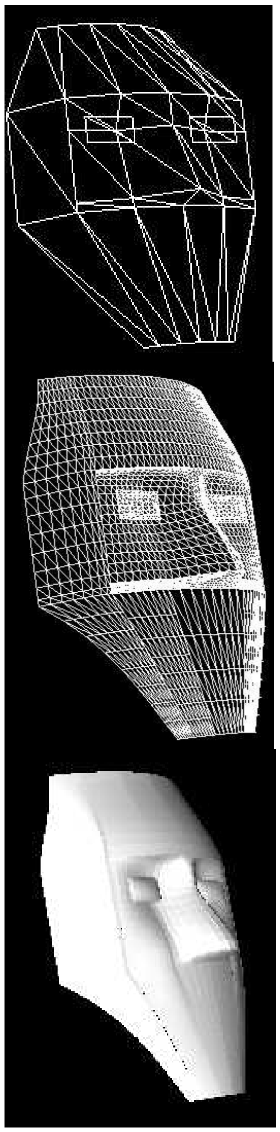
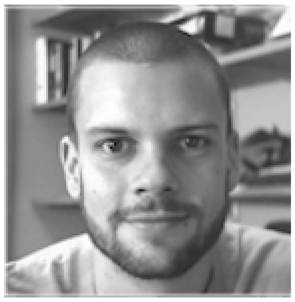


Figure 3: GazeMaster head model. Top: Bezier patches with no subdivision. Middle: patches subdivided. Middle: patches subdivided and shaded.

While modeling some curvature for the major head features to achieve occlusion is important, modeling minor features is not. Shadows and creases in the face will be present in the video already, and will be perceived as depth information. Therefore, our head model consists of an egg-like head shape, with a protrusion for the nose, indentation at the eye sockets, and outward curvature for the eyes themselves (see Figure 3).

For eyeball synthesis, we have demonstrated that even very simple planar models inside the segmented eye area can be very convincing, as shown in Figure 4: (a) Original image (b) eyes replaced with planar modelFigure 4 [1]. However, as discussed below, we have found eye segmentation to be very challenging. Therefore, our current prototype does not synthesize the eye; it merely provides some outward curvature to the eyes and texture maps the eyes from the video. This method has been able to achieve gaze-adjustment, but is not totally satisfying in regards to establishing eye-contact (see below). Note, however, that eyeball synthesis would be primarily used for horizontal gaze adjustments only, as vertical adjustments lead to changes in facial expression (see [1] for an explanation).


(a)

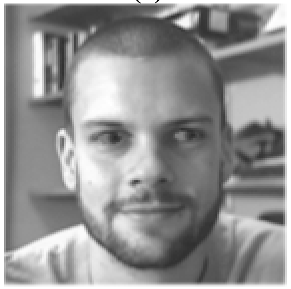

(b)

Figure 4: (a) Original image (b) eyes replaced with planar model

The GazeMaster head model is constructed mathematically from the spacing of the feature points that we track on the face, which include the eye corners, and the nostrils. Constants are multiplied by the eye-spacing, or eye-nostril spacing to estimate values for the head width at the forehead and chin, head height, and the width and depth of the nose. Bezier patches are used for each region of head. For example, the corner of one patch at the corner of the eye is defined as:

```
(LeftEye.LeftCorner.x - EyeGapX/4.0,
LeftEye.LeftCorner.y - EyeGapX/2.0, CIRC(fLeftX))
```

Where CIRC is a macro the finds points on a circle with a predefined radius that we are using as the shape across the brow.

The constants that we use require slight adjustment for different heads. Therefore, a persons head is represented in GazeMaster by a file containing the location of the feature points in a head-on view, along with the constants. The per-person initialization step required to acquire this data simply involves running the vision software on a head-on image, and "nudging" the constants until a satisfactory head model is produced.

Manipulation of the head orientation with our head model works quite well (Figure 7 shows the original video along with the head model tilted downwards by 17 degrees). However, we have found that the modeling around the eyes is very sensitive, and any inaccuracies in the model or the texture mapping (due to inaccuracies in the vision information) may yield undesired gaze adjustments. For example, see Figure 5. At top, the head is rotated downward by 21 degrees. At bottom, the rotation is increased to 23 degrees. Notice, however, that the left eye actually appears to be looking further *upward.* Close examination of the top of the left eye in the bottom image shows how the eye has been mapped too high on the eye in the model to give this effect. Improvements in the head model, and in eye tracking may be able to solve this. It may also be helpful to track the pupil locations, and consider what gaze direction will be achieved by mapping the pupil on to different parts of the eye-model curve.

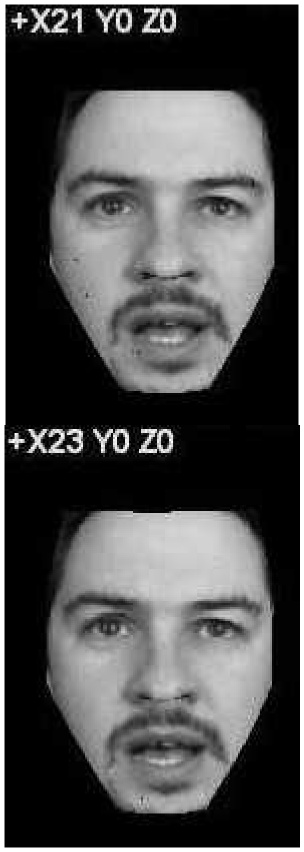


Figure 5: Eyes are very sensitive to the model and their mapping. At top, the head is rotated downward by 21 degrees. At bottom, the rotation is increased to 23 degrees. Notice, however, that the left eye actually appears to be looking further *upward.*

Currently, our model is just of the face area. It would be nice to have the rest of the head, and even the person's body. One approach would be to have a static full head model with our face superimposed. If the change in head orientation is small enough, a quick and dirty solution is to simply show the head model in front of the original video, as illustrated in Figure 6. In this case, the head model cannot extend as high into the forehead, or it will be seen sticking above the original video head as it rotates downward.

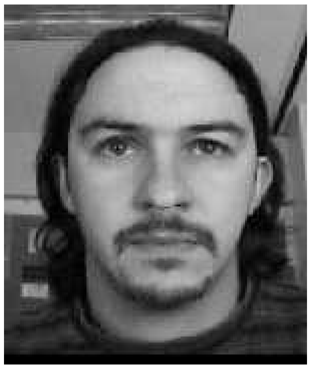


Figure 6: Face model superimposed on original video to recover the rest of the head and the body. The model has a downward rotation of 17 degrees. We shortened the forehead to avoid having the top corners of the model sticking above the original video head.

Debugging Support

In order to debug a system like GazeMaster, it is necessary to include many features in the software that wouldn't be needed in a videoconferencing product:

- Recorded material must be supported, to allow problem sequences to be exactly repeated with different versions of the code. Pause, play, and stepping through the video frames are necessary to allow examination of particular problem video frames. The software should support a file source both in place of a network reception, and in place of a camera source.

- 3D zoom and pan is needed to support close inspection of particular features on the face.

- To debug the head model, it must be viewable in wire frame and shaded modes, in addition to having the video texture-mapped on it. We found it most helpful to show the shaded model, the texture-mapped model, and the raw video simultaneously (see Figure 7).

- The orientation of the head model must be fully adjustable so that it can be viewed from different angles.

- The vision tracking points may need to be visible to determine if tracking inaccuracies are the source of a problem (in Figure 7 they are marked with blue crosses).

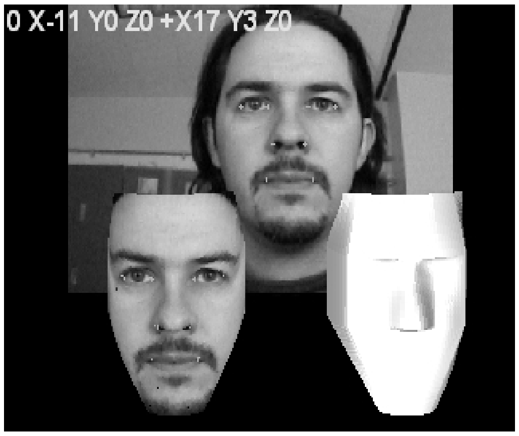


Figure 7: Debugging with raw video, texture-mapped head model, and shaded head-model. The frame number, orientation of the head, and adjustment of orientation is displayed in the upper left corner. Vision tracking points are marked with blue crosses.

Conclusion

Lack of Gaze awareness is one critical failing that prevents widespread acceptance of videoconferencing. Another is lack of ubiquity, so we have sought a solution that can be a software upgrade to a PC with a single camera. We have described our software architecture, and explained our approach to head modeling and debugging support. Our computer vision technology, while promising, is still immature. However, we are pleased with our results with respect to video, graphics, and networking. With additional work, we believe that gaze-aware videoconferencing will become a reality.

Acknowledgements

We have had many collaborators on the GazeMaster project: Tyler Beam, Thomas Kang, Volker Krueger, Steven Seitz, Kentaro Toyama, and Larry Zitnick. Gordon Bell, Jim Gray, and Roger Lueder helped evaluate our prototypes and made many excellent suggestions.

References

[1] Gemmell, Jim, Zitnick, C. Lawrence, Kang, Thomas, Toyama, Kentaro, and Seitz, Steven, Gaze-awareness for Videoconferencing: A Software Approach, *IEEE Multimedia*, 7(4), Oct-Dec 2000, 26-35.

[2] Argyle, Michael, *Bodily Communication* (Madison, Connecticut, International Universities Press, 1988).

[3] Novick, David G., Hansen, David G., Ward, Karen, Coordinating turn-taking with gaze, *Proceedings of the International Conference on*

Spoken Language Processing (ICSLP'96), Philadelphia, PA, October, 1996, 188-191.

[4] Yang Ruigang and Zhang, Zhengyou, Model-based Head Pose Tracking With Stereovision, *The 5th International Conference on Automatic Face and Gesture Recognition (FG02),* May 20-21, 2002 Washington D.C., USA.

[5] KentaroToyama and G. Hager, Incremental Focus of Attention for Robust Vision-Based Tracking, *International Journal of Computer Vision*, 35(1), 1999, 45-63.

[6] Kang, Thomas, Gemmell, Jim, Toyama, Kentaro, A Warp-Based Feature Tracker, *Microsoft Research Technical Report*, MSR-TR-99-80, October 1999.

[7] Feris, Rogério Schmidt, Gemmell, Jim, Toyama, Kentaro, and Krueger, Volker, Hierarchical Wavelet Networks for Facial Feature Localization, *The 5th International Conference on Automatic Face and Gesture Recognition (FG02)* May 20-21, 2002 Washington D.C., USA.