Real-time Augmented Reality With IGSTK

Z. R. Bárdosi[1], W. Freysinger[1]

[1] HNO Klinik, Medizinische Universität, Innsbruck, Österreich

Contact:zoltan.bardosi@i-med.ac.at,
wolfgang.freysinger@i-med.ac.at

Abstract:

The IGSTK (Image-Guided Surgery Toolkit) library is a free and open toolkit for the development of prototype surgical navigation applications. IGSTK contains all necessary modules required for surgical 2D and 3D visualization, DICOM data import, various 3D-tracking systems and handling of live-video input directly or over a network.
IGSTK was added a new component for real-time, interactive augmented reality visualization, IGSTK-AR, that can handle conventional and wide-angle cameras, provides for camera calibration, camera-to-tracker registration, low-latency processing of input video streams, and GPU based low-latency overlay visualization.
The implementation was tested with a 4 mm endoscope and an ordinary camcorder (Canon MV 20) and had a system latency that is useful for surgical application, 272.8 ± 25.6 ms (min: 240 ms, max: 320 ms). The system was run with a non-optimized code.
IGTK-AR seamlessly integrates to the current IGSTK View architecture and yields a fully object-oriented IGSTK system architecture.

Keywords: IGSTK, augmented reality, real-time video processing

1 Problem

Developing CAS (computer assisted surgery) systems is challenging due to the complexity of the computation tasks involved, and, specific to surgical applications, due to the need for robustness and high-quality usability. CAS systems need medically certified hardware requiring specific skills from developers and users; an iterative development cycle is typical. Research in this field typically requires rapid development of customized, robust prototypes that ease the medico-legal certification process what is met by open source devices and code. Commercially available systems are built to purpose, expensive, and proprietary, all of which impedes research and development in academic environments. In the field of CAS there is a variety of free and open-source software packages (e. g. ctk.org, mitk.org, slicer.org), and IGSTK.org [1,2]. IGSTK is a cross-platform open-source C++ framework offering a well-structured, state-machine driven internal architecture, supports the most commonly used 2D and 3D visualization methods, point-based registration, a wide variety of 3D-tracker devices and can handle video input. It comes with a set of advanced examples, some of which are approved for clinical use.
IGSTK v. 5.0 does neither provide low-latency real-time video stream processing nor augmented reality (AR), even though these areas are hot topics in current CAS research. To allow a clinical use, the system response time has to be minimal to have no noticeable latency in the control feedback loop. We present extensions to IGSTK in a generic way for rapid development of clinically usable AR-CAS systems based on this platform.

2 Methods

AR visualization in IGSTK is accessible as a set of new classes derived from the generic igstk::View class that can visualize standard scenes: anything that can be visualized in the standard 2D/3D views can be visualized in the AR overlay, see Fig. 1. The rendering of the AR views is a GPU accelerated two-step process. The first step renders the undistorted 3D-scene with a special camera setup into an offline texture buffer which is then combined with the current video frame in real-time. The sampling in the second step is also responsible for the distortion correction.

A typical hardware setup for AR applications consists of a camera, 3D-tracking, a patient and 3D medical data. Physical scene objects (tools, patient, etc.) are tracked and registered to 3D patient data. Perioperatively, an augmented version of the video from the surgical site is presented to the surgeon, what requires the simultaneous handling of video and 3D-tracker data streams. IGSTK needs to be extended by modules for camera calibration; camera-to-tracker calibration; low-latency video input processing and overlay visualization.

Camera calibration provides estimates for the optical parameters of the camera use (i.e. lense distortion, focal length, etc.) used and is crucial to fuse video images with 3D navigation structures. Camera calibration, Kannala-Brandt [3], supporting conventional and wide-angle optics, is implemented as a stand-alone module, with a planar 3D calibration pattern (8x7 circular blobs), that is simultaneously tracked in 3D. Once the intrinsic camera parameters are known, pixel-wise undistortion maps from camera images to undistorted projections of the 3D scene are calculated by forward-projecting every pixel. The undistorted image resolution and field-of-view are estimated from the camera images. This calculation is highly parallelizable, amenable to considerable speedup. The extrinsic camera parameters relative to the 3D-tracker are estimated with the traced calibration pattern and the Direct Linear Transformation [6]. For tracked cameras, the transformation from the camera's rigid body to its optical center is determined. All parameters are exported in a configuration file as input to the IGSTK augmentation views.

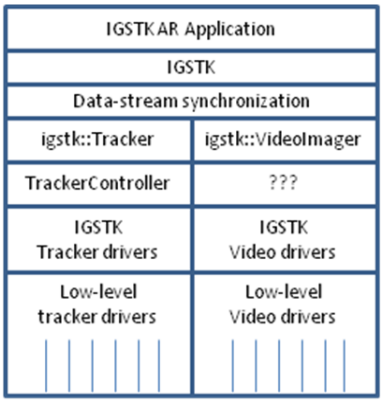

Figure 1: Standard IGSTK AR architecture

A new video processing concept, the SharedMemoryVideoManager in IGSTK, was created to speed up processing and to reduce latency, see Fig. 2. This can be done with popular GUI toolkits (e.g. Qt, nokia.com) or directly via the operating system's API. Access to the shared memory buffer is controlled with the standard (semaphore based) thread-safe methodology. The SharedMemoryVideoImager assumes that low-level video drivers are separated into different applications. A small, external cross-platform OpenCV (opencv.org) „video broadcaster" was written to read and send video frames to the shared memory video buffer, from where they are directly uploaded to GPU memory by the AR view class. IGSTK-AR applications so only depend on a generic video frame reader using a shared memory object to access the frames. Video data are read only once, when being uploaded in the AR View class to the GPU. IGSTK was further extended with an OpenIGTLink [4] based tracker class, that receives tracker data via an OpenIGTLink client-server connection, see Fig. 2. The OpenIGTLinkBroadcaster application provides any IGSTK supported tracker over TCP/IP network.

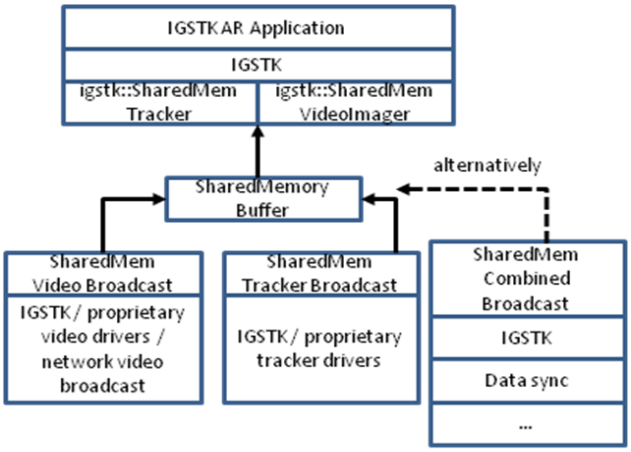

Figure 2: Shared memory IGSTK AR architecture

A demonstrator AR application, using the OpenIGTLink tracker and the new AR view classes from the IGSTK-AR, was written. It reads the descriptions of virtual objects that are connected to tracker tools from an XML file, which defines the location of the mesh data files, color and opacity for the rendering, name of attached tracker tool and registration parameters of the object to the tracker tool. On base of these objects an ordinary scene-graph is built and visualized in an augmented video overlay view. The tracker was completely handled by a separate application, the OpenIGTLinkTrackerBroadcaster example, which pumps the tracker data into an outgoing OpenIGTLink connection; this TCP/IP stream was received by the new OpenIGTLinkTracker module in the AR application. The AR demo used an off-the-shelf camcorder (Canon MV 20, DV) as video input, and an optical tracking system (CamBar B2, Axios 3D [5]). The camcorder was mounted on the stand of the tracking device so that the tracker's center of the measurement volume was aligned with the center of the image taken by the video camera. The video signals (DV, 756x576, RGB) entered the PC via a firewire card. The AR application and the two broadcaster applications were running on a high-end PC setup (an Intel i7 3.4Ghz CPU with 4 cores, a GeForce GTX 460 GPU and 8 gigabytes of RAM, Windows 7, DirectShow interface for openCV). The CamBar B2 tracker requires a high-end machine to calculate the poses from the tracker camera stereo images the Cambar B2 sends via GigE to a proprietary application.

Optical calibration was done with approximately 20 different images of the calibration pattern; images and 3D-positions of the attached tracker tool were logged. The views were analyzed automatically with OpenCV's circular grid detector, camera calibration and camera-to-tracker registration were executed and the parameters stored.

CT images (1 mm slice thickness) from a plastic skull passive tracker markers and implanted titanium screws were used to build a 3D model of the maxilla (marching cubes algorithm and decimation to 20 % of the original triangle count) and the whole head; the registration of the maxilla to the rest of the skull was calculated. These segmentations were visualized with the AR overlay application.

3 Results and Discussion

The skull was rigidly registered to the CT-data using the screws with < 1 mm RMS, which is a sufficient application accuracy. Pixel-wise distortion [and undistortion] between camera images and undistorted projections of the 3D scene are calculated by forward- [and back-]projecting every pixel in less than 1 minute on one core of a recent PC. the distance between the camera (+ tracker) and the object was approximately 1 meter, and the used camera had less than 60 degrees field-of-view. This distance has corresponded to the near border of the optimal working volume of the optical tracking device. The precision of the overlays in the video of the DV-videos were visually evaluated by overlaying a virtual coordinate-system on the tip of the pointer-tool at different positions. The overlay error was found to be around 1 mm, which was adequate for the demo application.

To evaluate the speed performance of the system, a feedback loop was created by filming a plastic skull (with retro-reflective markers) and the screen. The PC screen showed a digital watch with centisecond resolution and the augmentation. This feedback loop generates a series of past frames as part of the background and allowed to measure the latency of the system. Photographs were taken of the monitor at various times, the differences between the time displayed by the stopwatch application and the time-stamp visible on the processed AR-image were evaluated. 80 images were randomly sampled from a twenty minute long sequence. The latency was found to be 272.8 ± 25.6 ms (min: 240 ms, max: 320 ms). This includes all processing and rendering steps. From this performance it is concluded that the proposed modifications provide a low-latency real-time augmented reality module for IGSTK. It can be used for rapid development of prototype applications in the AR field. The performance of the rendering system runs without noticeable latency even when using FullHD resolutions. The precision of the calibration is comparable to or better than other widely used methods [7]. The distortion map's structure and the intrinsic parameters are generic, so it can be created with any suitable camera calibration method.
Compared to the standard video imager class, the structure was greatly simplified (no ring buffer and device-dependent drivers). This reduces the processing load and complexity on the application side, eases interfacing with proprietary (video) drivers and allows device independent video handling and network video streaming inside IGSTK. Video and tracker synchronization can be done with minimal effort.

Figure 3: Photo of the AR-IGSTK demonstrator latency measurement: A sample measurement which shows a lag of 240 ms.

The abstraction of the video drivers, Fig. 2, has major advantages over the current integrated VideoImager architecture, Fig. 1: device-independency on the receiver side of the application, and increased flexibility to run with different hardware without recompiling. Moreover, proprietary drivers do not have to be integrated into IGSTK: a small application that drives the hardware and writes to the shared memory buffer is enough.
The new approach (see Figure 2) simplifies the application's portability and internal structure: no link to untested or proprietary video drivers, and no low-level processing of acquired data. A considerable speed-up is due to the fact that IGSTK does not have to handle intermediate video data internally. This architecture eases software verification and avoids the need to keep track of low-level driver dependencies in the main application. Additionally, the number of possible states within the application explodes with the increasing number of support libraries handling different devices. Now, the main application contains only a few simple device-independent drivers. Medical certification also becomes easier, since it is enough to prove once, that the main application is correct, and that it handles any possible data that appears in the shared memory buffer properly. In case of extensions, it is enough to

prove that the new modules as separate units fill the shared memory buffer with correct data.
The IGSTK-AR architecture also makes it simple to add features like network video-streaming and playback capabilities into navigation. The shared memory architecture can also be used to store video frame(s) **and** other information like synchronized tracking data stream. A SharedMemoryTracker can read the state of the tracker tools from this buffer and an eventual synchronization of video and poses is safely maintained inside IGSTK, without time-consuming frame ID/timestamp matching.
The new architecture enhances usability and clinical reliability as now not the whole application will crash: if the driver application crashes, the main application continues, and it can report that the data in the shared memory is outdated or invalid. Restarting the small feeder application does not break the whole workflow, and avoids data loss.
The abstraction and separation of the tracker and video drivers from the main application overcomes the lack of a device-independent „TrackerController“, which can initialize every IGSTK-supported tracker or video device without recompiling the whole application: devices can be changed without recompiling or even without restarting the main application. This can be a crucial advantage in a clinical scenario, when e.g. due to a hardware failure a device needs to be exhanged intraoperatively.
The OpenIGTLink can run on a dedicated PC (minimum a dual-core CPU) and transfer the data with the OpenIGTLink two the visualization PCs to distribute the computational load. In this configuration, the AR system only requires one CPU core and a recent GPU with pixel-shader support. The demo AR application was run successfully with the NDI Vicra and the CamBar B2 trackers and with various full HD video camcorders in this split configuration.

4 Acknowledgement

This work was funded by the Jubilee Fund of the Austrian National Bank, grant number 13 003.

5 References

[1] Cleary K., Enquobahrie A., Yaniv Z., Ibanez L., Aylward S., Zhang H., Gobbi D., Jomier J., Kim H-S., Cheng P., Blake B., Gary K., *IGSTK: The Book.* Signature Book Printing, Gaithersburg, Maryland, USA,. 2007.
[2] http://www.igstk.org
[3] J. Kannala, S. S. Brandt, *A Generic Camera Model and Calibration Method for Conventional, Wide-Angle, and Fish-Eye Lenses*, IEEE Trans- actions on Pattern Analysis and Machine Intelligence, **28(8),** 1335- 1340, 2006.
[4] http://www.na-mic.org/Wiki/index.php/OpenIGTLink
[5] http://www.axios3d.de
[6] Abdel-Aziz,Y.I., Karara,H.M., *Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry*, Symp. Close-range Photogrammetry, 1 – 18, ASP Symposium on Close-Range-Photogrammetry, 1971
[7] Tsai, R. Y., *A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses*, IEEE J Rob. Autom **RA-s(4)**, 323-344, 1987.