LightTracker: An open source multi-touch toolkit

Adam Gokcezade, Jakob Leitner, Michael Haller
Media Interaction Lab
Upper Austria University of Applied Sciences
Hagenberg, Austria
mi-lab@fh-hagenberg.at

ABSTRACT
In this article we present LightTracker – an open source toolkit for vision-based multi-touch setups. Using this toolkit, we can dynamically create and manipulate the image processing pipeline at runtime. After presenting various new requirements derived from hardware configurations as well as literature review, features and shortcomings of available tracking solutions are discussed and compared to our proposed toolkit. This is followed by a detailed description of the toolkits functionality including the filter chain and the improved calibration module. We also present implementation details such as the plugin system and the multi-threaded architecture. To illustrate the advantages of the toolkit, an interactive gaming couch table based on LightTracker is introduced.

Categories and Subject Descriptors
D.5.2. [**Information Interfaces**]: User Interfaces—*Input devices and strategies; graphical user interfaces; prototyping*. D.2.2 [**Software Engineering**]: Design Tools and Techniques— *User interfaces*.

Keywords
Multi-touch, Interactive Surfaces, Computer Vision, Tabletop, Toolkit

1. INTRODUCTION
Interactive multi-touch displays and tables have attracted a lot of attention over the past years and are increasingly becoming popular in different domains including gaming [2][3] [8][16][17][18], entertainment, and museum/art installations [24]. Currently, there are two possibilities to start with the development of tabletop applications. One is to get an all-in-one solution, where several companies are providing hard- and software solutions [20]. Very often, these solutions are not flexible enough. The rendering PC (Intel Core 2 Duo @ 2.13 GHz) of the Microsoft Surface, for example, cannot be upgraded easily. Moreover, the table itself is not designed to be customized.

Alternatively, do it yourself (DIY) multi-touch tables based on optical tracking are increasingly becoming interesting. Hardware solutions based on Frustrated Total Internal Reflection (FTIR) [11] and Diffused Illumination (DI) [19] have enabled the low-cost development of such surfaces. Several online communities and discussion groups provide enough details for the development of vision-based multi-touch setups. In addition, different community-driven open source tracking solutions have emerged.

These tracking solutions provide an excellent starting point, since they offer enough functionality for simpler setups. Most suffer from a monolithic design tailored to the most common hardware setups and provide only little variation by tweaking certain parameters. While developing non-standard multi-touch setups, we found ourselves spending a lot of time trying to extend solutions like Touchlib [28] and CCV [5] to fulfill our requirements. Due to the inflexible design of these trackers this proved to be more complicated than we had thought.

From this experience, we derived a list of requirements, which an ideal tracking solution should satisfy:

- Run-time setup of a custom image-processing pipeline using a modular system,
- Clean API for easy extensions in the form of shared libraries,
- Support for Multicore-CPUs,
- Support for touch and marker tracking in one combined pipeline,
- Multiple camera support, and
- Improved support for setups using wide-angle lenses.

In this paper, we present LightTracker, a modular and open source multi-touch toolkit. LightTracker addresses all of these requirements by providing an easy-to-use toolkit with a highly-flexible and multi-threaded image-processing pipeline, an improved calibration procedure and a strong focus on extensibility.

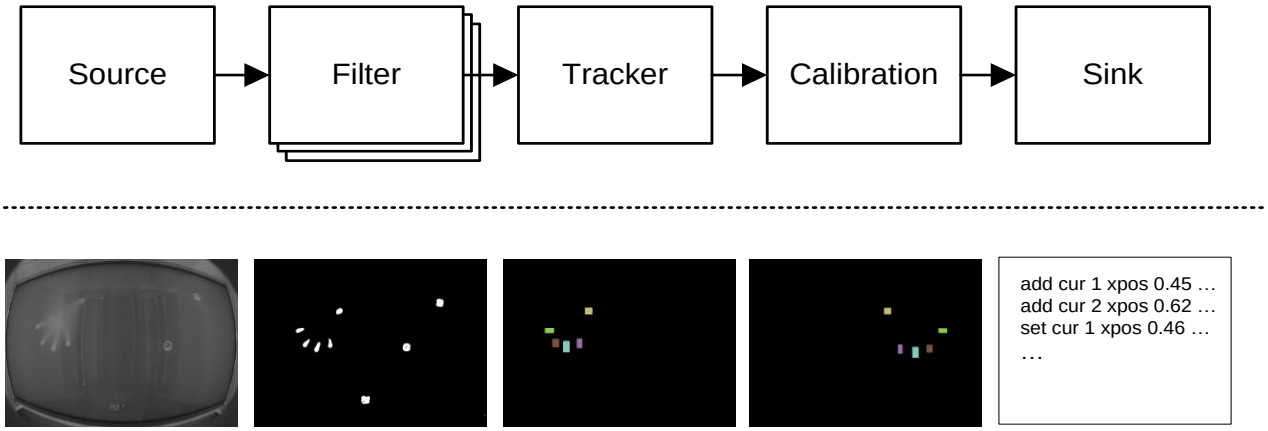


Figure 1: A schematic view of the image processing pipeline showing example output at each stage.

2. RELATED WORK

As proposed by Echtler et al., we can divide the multi-touch libraries and toolkits into two groups: low-level input processing tools and high-level interaction software [7].

2.1 Low-level input processing tools

Touchlib was one of the first vision-based open source multi-touch solutions created for the Microsoft Windows platform. The library itself is linked directly into the multi-touch application. The image processing pipeline can be tweaked by editing a XML configuration file. This provides a certain degree of flexibility but changes in the pipeline cannot be performed at runtime, but need a restart of the application. It also lacks a proper GUI.

BBTouch is another open-source multi-touch library implemented with Cocoa and Objective C and therefore only available for MacOS systems [1]. The BBTouch platform offers a compact GUI for tweaking different parameters. It also offers a unique calibration method in comparison with all other open source tracking solutions but provides no means for editing the filter pipeline as with Touchlib.

Touchè is another tracking library primarily designed for the MacOS system [27]. In contrast to other trackers, Touchè comes with a mature graphical user interface, which allows programmers to change a lot of parameters of the different filters in the image processing pipeline but operates also on a fixed filter pipeline.

Community Core Vision (CCV) is a popular open-source and cross-platform solution for multi-touch tracking [5]. CCV supports image processing on the GPU. Newest builds also support multiple camera setups. As with most other solutions it comes with a fixed image processing pipeline and doesn't take advantage of multicore systems.

A newer project called movid (Modular Open Vision Interaction Daemon) takes a different approach by providing the dynamic creation of filter graphs using a node-based interface [21]. These graphs can be created using a browser. It's also the only solution which aims at combining touch and marker tracking as well as supporting image processing using more than one thread. movid is, at the time of this writing, not available as a binary as a first official release is still under development.

ReacTIVision is another open-source computer vision library used for multi-touch setups [14]. In contrast to the aforementioned tools it was primarily designed for table-based tangible user interface and is also able to track fiducial markers. It offers a limited GUI and no mechanism to adapt the image processing pipeline to nonstandard setups.

2.2 High-level interaction libraries

To develop applications which take advantage of the multi-touch input generated by the aforementioned low-level input processing tools several libraries and toolkits exist.

Commercial tabletops are usually shipped together with a high-level multi-touch library. One of the first tabletop solutions was the DiamondTouch [6], a non-optical table, which came with the high-level SDK DiamondSpin [26].

The Microsoft Surface SDK is another high-level multi-touch library, which works smoothly with the Microsoft Surface. The combination with the Windows Presentation Foundation (WPF) and XNA provides a rapid prototyping of games and installations. The Microsoft Surface SDK is limited to the Surface hardware.

Squidy is an interaction library, which unifies various device drivers, frameworks and tracking toolkits in one common library and provides a visual design environment [15]. A corresponding visual user interface hides the complexity of the technical implementation by providing a simple visual data flow programming interface.

PyMT [12], is a Python-based multi-touch environment, which allows users to prototype faster multi-touch applications. By providing a variety of multi-touch widgets, and easy to access advanced features, programmers can quickly write tabletop applications. Finally, there is also MT4J [22], which in contrast to PyMT, is based on Java.

Using this classification into low-level and high-level libraries, LightTracker falls into the first category improving on some of the aforementioned shortcomings of other solutions.

3. LIGHTTRACKER

With LightTracker, users can dynamically create and manipulate the image-processing pipeline at run-time (cf. Figure 1). The toolkit is primarily designed for touch tracking, but the pipeline can easily be extended for other image processing tasks, for example marker tracking.

Figure 2: The four plugin sections of the LightTracker: The image created by the source (A) is processed by the filter chain (B), analyzed by the tracker (C), and the resulting blobs, after conversion by the calibration, are forwarded to the application by the sink (D).

Users are able to arrange the image processing pipeline by choosing individual modules from a pool of plugins, including:

- At the beginning of the pipeline there is a single **source plugin (**Figure 2 (A)). This plugin is responsible for providing image data for the pipeline. Possible implementations include simple camera sources, video players or plugins which stich together multiple image-sources into one combined image.
- The source plugin passes the source image to the first **filter plugin** of the filter chain **(**Figure 2 (B)). These plugins perform image processing and pass the resulting image to the next filter of the chain. Examples include background subtraction filters, threshold filters or more complex functionality like marker tracking. The screenshot in Figure 2 (B) shows a pipeline with four different filters.
- The last filter in the chain passes the resulting image to a **tracker plugin (**Figure 2 (C)). This plugin takes the image and performs blob tracking. The result of the tracker plugin is a list of tracked blobs.
- The tracker plugin passes this list to the **calibration module** of the pipeline, which transforms the blob values from camera coordinates into screen coordinates. The calibration module is not visible in Figure 2 as it has its own UI and is explained in detail in Section 3.2.
- After the calibration step, the blobs are forwarded to the **sink plugin (**Figure 2 (D)). This plugin is the last in the chain and responsible for encoding and sending the blob data to an application. Possible implementations include a TUIO [13] sink plugin encoding the touch data in the popular TUIO protocol and sending the data to the client over UDP.

The resulting image processing pipelines can be saved using a XML file format. By loading different pipeline setups, we can quickly switch between differently specialized pipelines. This allows us to define and manage different pipelines for touch-tracking only as well as a combined marker and touch-tracking pipelines for the same hardware setup.

3.1 Filter Chain

The filter chain is at the heart of every touch tracking solution. This is where the image from the source is processed in a way that good blob tracking results can be achieved. Consequently, this is where the most user customization is required.

With the exception of movid [21], all existing open source touch tracking solutions provide a *fixed* filter pipeline with a standard set of filters which cannot be rearranged at runtime. Customization is only achieved by tweaking parameters of the individual filter steps. In contrast, LightTracker's filter chain allows users to assemble a chain of image processing filters of arbitrary length. Filters can be added, removed and re-ordered dynamically at run-time. The result of each modification can be observed immediately in the preview of each filter. Consequently, we can prototype filter chain ideas more easily as well as offer a lot more flexibility when building non-standard multi-touch setups.

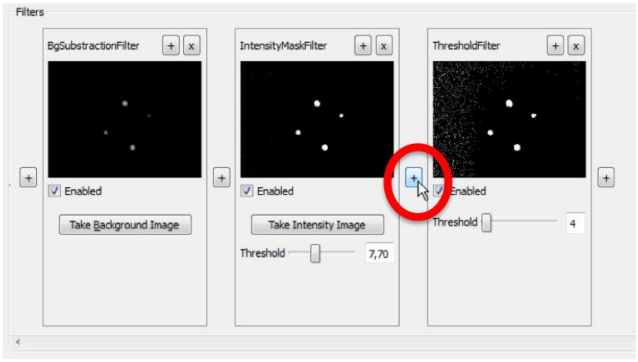


Figure 3: Users can dynamically add new filters by clicking the '+' button displayed at all possible locations in the filter chain.

In comparison to the graph-based approach of movid, which provides branching of the filter pipeline, the single filter chain approach of LightTracker offers less flexibility. On the other hand no image copying operations between filter steps are required.

To enable a more detailed view of the resulting image, single plugins can be maximized, as well as minimized to free space for other plugins, aiding with the fine tuning and debugging of the filter chain.

By clicking one of the '+' buttons in between filters, a list of all available filter plugins is displayed (cf. Figure 3). By selecting a plugin from the list, it is inserted into the pipeline (cf. Figure 4). Removing a filter from the chain is achieved by pressing the 'x' labeled button located in the upper right corner of each filter.

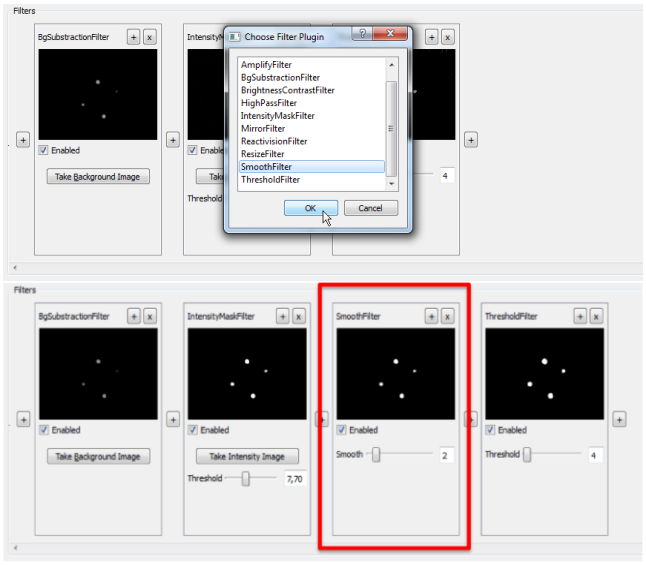


Figure 4: Inserting a filter into the pipeline.

3.2 Calibration

LightTracker's calibration process is similar to that of other tracking applications: In a multi-screen environment, users first have to choose a screen to calibrate. On the chosen screen a borderless full screen window is displayed showing a configurable clipping rectangle. This rectangle defines the area in which touches are "accepted". Inside this clipping rectangle a configurable amount of points is arranged. When starting the calibration process the user has to touch these points on the screen one after the other. This results in point pairs linking camera coordinates (the touches in the camera image) with screen coordinates (the screen coordinates of the points). By grouping adjacent point pairs into quads we can calculate the perspective transformation which links the quad in camera coordinates with the quad in screen coordinates. Therefore, enabling us to translate any point in camera coordinates inside a quad, into the corresponding point in screen coordinates.

This transformation delivers "correct" values only under the assumption that the camera image does not suffer from nonlinear distortions. Otherwise touch points which do not directly lie on the calibration points will deviate from the expected position in screen space. This problem can be observed when using an ultra-wide angle camera lens such as a fisheye lens which suffers from high amounts of barrel distortion. With other tracking solutions the magnitude of this positional error can be reduced by choosing a higher amount of calibration points. However this in turn makes calibration a longer and more tedious process.

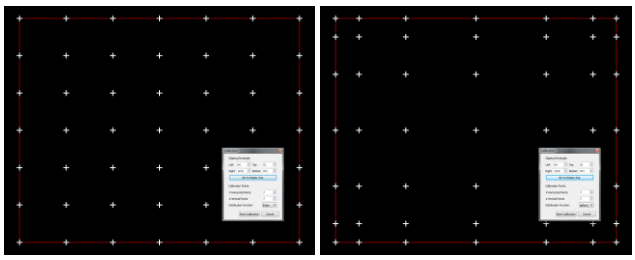


Figure 5: Linear *(left)* and spherical distribution *(right)* of calibration points.

To improve on this, LightTracker offers different distribution functions for the calibration points (Figure 5). By distributing the points in a nonlinear fashion, more calibration points are provided only in areas where a high amount of nonlinear distortion is present. For fisheye lenses this is the case in the border regions of the image. Through this it is possible to get better precision in areas where needed (see Figure 6, *left*) while resulting in a shorter overall calibration time. At the moment only a linear and a spherical distribution function is implemented but additional functions can be added easily.

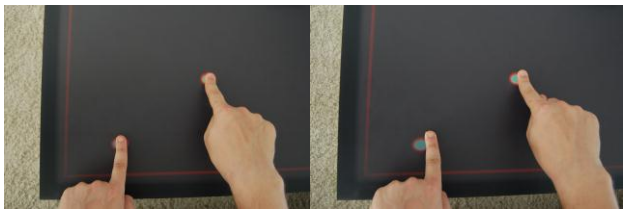
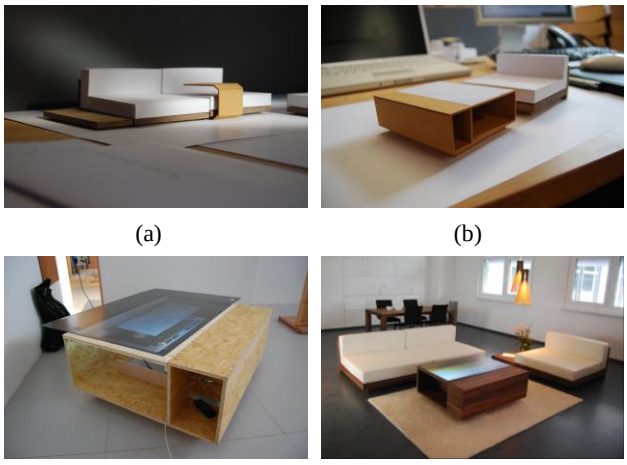


Figure 6: Due to nonlinear distortions in the border regions a spherical distribution of calibration points *(left)* yields better results than a linear distribution *(right)* while using the same amount of calibration points.

4. APPLICATION CASE: GAMING COUCHTABLE

To demonstrate the performance of LightTracker, we implemented a tabletop multi-touch setup based on DI tracking. The main goal was to create an interactive couch table for playing games which would also function well as a normal piece of furniture in an ordinary living space under conditions such as changing ambient lighting. Figure 7 illustrates a couple of design steps of the table.



(a) (b)

(c) (d)

Figure 7: Starting from first paper mockups of possible tables (a) and the final design (b) a first full size prototype was built (c). The last image (d) shows the final table.

The various design iterations will not be discussed in this paper. Instead, we will focus on how details of the final setup influenced the configuration of the image processing pipeline using LightTracker.

4.1 Hardware

In comparison to other multi-touch tables, the most outstanding feature of our couch table is its very low height in relation to the size of the screen. The table is only about 15 inches high measuring a screen diagonal of 46 inches. Adjacent to the screen, and of the same height, there is a wooden sideboard, which can be used as additional space for depositing objects. Under the sideboard there is enough space for the PC and the projector. On both sides of the table, beneath the screen, we added additional storage compartments for magazines.

The limited space inside the table is housing a short-throw projector (Hitachi CP-A100), the rendering PC, one high resolution camera (Pointgrey Flea 2) with a fisheye lens (185° FOV) as well as IR-LED strips for illumination.

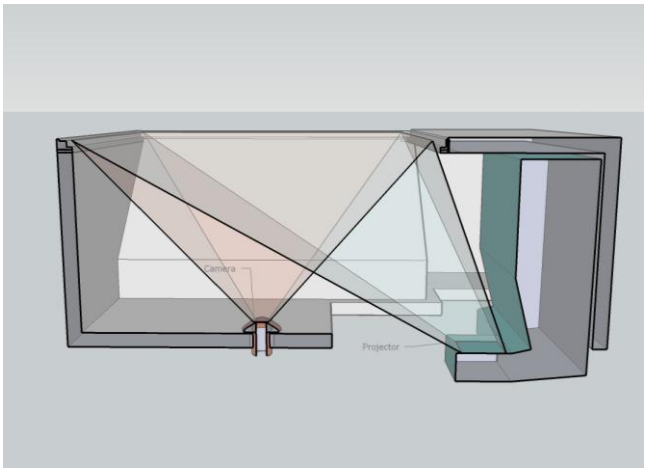


Figure 8: A camera with a fisheye lens in the middle of the table is used for tracking. To display the image a short-throw projector is used. In the background one of the magazine compartments can be seen constraining the space inside the table.

Due to the low height of the table and the size of the screen we had the choice of either using a single camera setup with a very wide angle lens or to arrange two or more cameras inside the table in such a way that each of them covers only a part of the screen. Because of the space restrictions imposed by the two magazine compartments we decided to use a single camera with a fisheye lens (cf. Figure 8).

Based on this hardware setup we observed two problems which cannot be solved easily with state-of-the-art multi-touch solutions:

- Distortion of the camera image due to fisheye lens.
- Uneven lighting especially in border regions due to limited space within the table.

The first problem can be solved using the nonlinear distribution of calibration points in LightTracker's calibration module (cf. Section 3.2). The solution to the second problem as well as the general setup of the filter chain is discussed in more detail in the following section.

4.2 Filter Chain

As a starting point for our filter chain setup, we looked at existing tracking solutions and their filter setups. The resulting image processing pipeline for the table looked as follows:

- Background subtraction filter,
- ReacTIVision marker tracking filter,
- High-pass filter,
- Blur filter, and
- Threshold filter.

This filter chain worked reasonably well for the center part of the screen. However in some regions blob tracking as well as marker tracking did not produce usable results. Uneven lighting inside the table as well as the vignetting effects of the camera lens resulted in a brightness falloff in the image for border areas of the screen. Consequently, blobs in that area could not be detected reliably.

Due to the special shape of the couch table an even light distribution inside the table is practically impossible. Likewise the vignetting effect of our fisheye lens can only be countered by using more expensive hardware. That is why we decided to approach this problem by adding a custom filter to the filter chain, the so called "Intensity Map" filter. It adjusts the brightness of the images on a per-pixel basis. Hence areas with darker blobs are getting lighted up to the same level as areas, which produce lighter blobs.

The filter requires a calibration step in which the intensity for each pixel in the camera image is determined when light is reflected at that position. This is done by capturing a reference image when the whole touch surface is covered with a material, which has comparable reflective properties as fingers or other objects later used for interacting with the table. In our case, we use a piece of cotton cloth. The calibration process and the resulting reference image for our setup can be seen in Figure 9.

Figure 9: The calibration process for the intensity map filter (*left*) and the resulting reference image illustrating the uneven lighting situation (*right*).

For every pixel in this reference image the filter calculates the factor with which it would have to multiply the value to get a fully lit pixel and stores this factor inside a texture. By multiplying the incoming images with this texture the filter selectively brightens up darker areas in these images (cf. Figure 10).

This can only be used for smaller variations in image brightness. When used for very dark areas the filter leads to high multiplication factors, which also results in a strong amplification of artifacts like image noise. Hence the filter cannot replace even lighting altogether but helps to achieve a more even brightness distribution in the image and therefore leads to a more stable blob tracking.

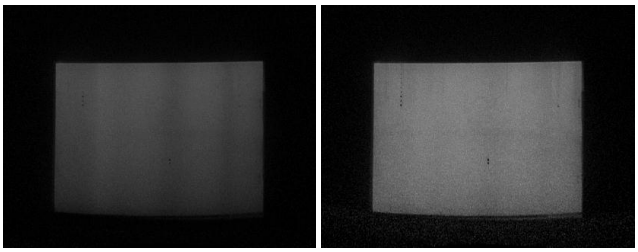


Figure 10: By putting a white sheet of paper on the table we can visualize the effect of the intensity map: On the left we see the unprocessed image showing uneven lighting (vertical stripes) as well as vignetting effects (brightness falloff at the bottom). The right image shows the processed image showing a more even brightness distribution.

The final image processing pipeline, for combined touch and marker tracking, including the "Intensity Map" filter is depicted in Figure 11. Each step offers the possibility to tweak certain parameters.

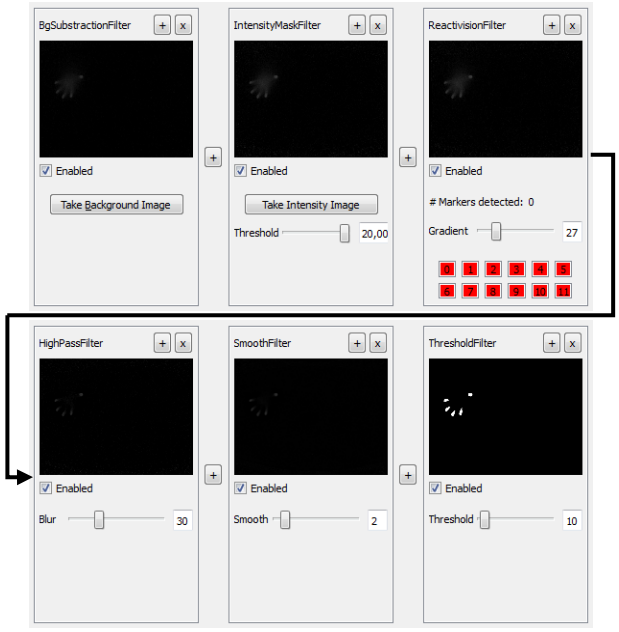


Figure 11: The final filter chain used for the table setup.

5. IMPLEMENTATION

LightTracker is implemented using C++, Qt [25] and OpenCV [23]. To provide more flexibility, every part of the pipeline – except for the calibration – is implemented using a plugin mechanism.

5.1 Plugin System

LightTracker already provides a set of common plugins and new ones can easily be created via an API. Each of the plugin types (source, filter, tracker, sink) has a base class with a lightweight interface from which new plugins can be derived. By compiling the derived plugin as a shared library and placing the library in a subdirectory, it is automatically loaded by LightTracker and ready to be inserted into the pipeline.

To facilitate the creation of custom plugins, our API already provides a lot of common functionality in the base classes of the plugin types. A minimal user interface providing a preview image of the plugin result, buttons to maximize, minimize and remove the plugin, as well as a checkbox to enable and disable the plugin, is automatically generated as part of every plugin. Developers can easily extend this user interface with custom controls for the specific functions of their plugin (cf. Figure 12).

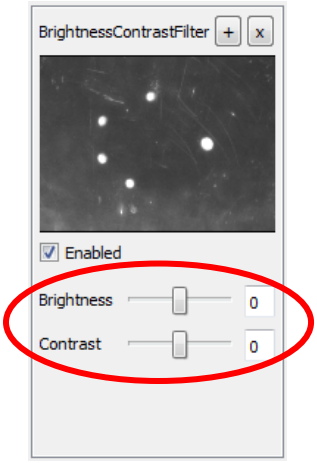


Figure 12: The graphical user interface of a simple filter plugin. The encircled controls have been added by the developer in addition to the standard controls from the base class.

Another feature inherited from the base classes is the property system. When creating a new plugin, users can save member variables in a property table. Consequently, these variables get serialized automatically once the whole pipeline is saved by the user as well as de-serialized when loaded.

5.2 Threading

Most open-source multi-touch trackers do not utilize the multicore architecture of modern computers by implementing the image processing pipeline in a single thread. The LightTracker-toolkit on the other hand was designed with a multi-threading architecture; thus enabling better performance on multicore systems.

Threading in LightTracker is implemented on a "*per-image*" basis. Rather than using a single thread for each plugin as for example implemented in movid [21], a single thread is responsible for executing the whole pipeline on a single frame. This minimizes the synchronization overhead between plugins.

One limitation of this frame-based threading approach is that plugins cannot easily access results from previous frames. For example, it is not possible for *frame(t)* to access the result of *frame(t-1)* if processing of this frame is still executing in a different thread. This limitation could be overcome by implementing a mechanism which would synchronize between different threads but this would have a negative impact on performance.

During our tests, however, we found that this limitation was an acceptable drawback for the performance gain and should the need for such a mechanism arise, this could easily be integrated into the current system at a later point in time.

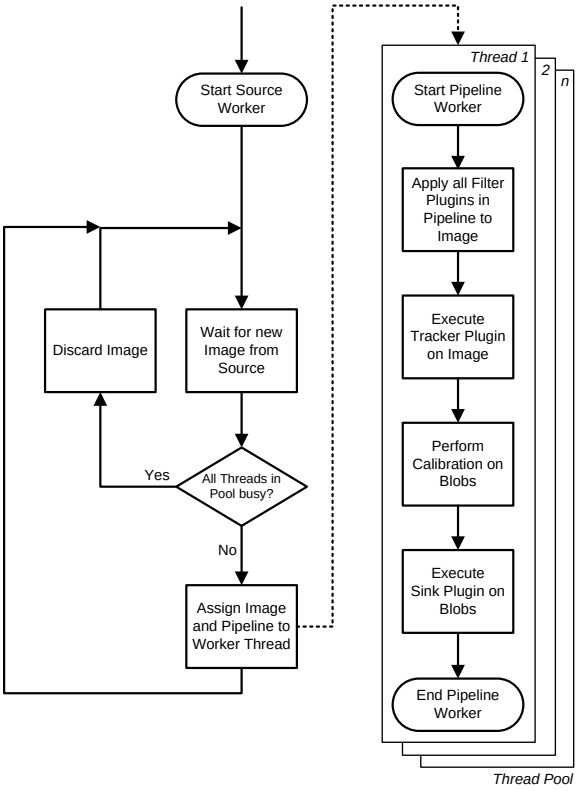


Figure 13: Multi-threading control flow in LightTracking: The SourceWorker thread is assigning created images to a PipelineWorker thread from the pool.

The multi-threading architecture of LightTracking Toolkit is implemented using two types of worker threads: SourceWorker and PipelineWorker (see Figure 13).

A single SourceWorker operates on the source plugin by checking in a continuous loop if a new image is ready to be processed. If an image is ready, the image and the current pipeline are assigned to a free PipelineWorker from the thread pool. Multiple PipelineWorkers process the images, applying the filter plugins, executing the tracker plugin, performing the calibration, and executing the sink plugin.

The number of PipelineWorker threads in the thread pool is initially calculated based of the real and logical number of processor cores in the system. This number can be adjusted in the application settings and represents the maximum number of images, which can be processed simultaneously. If the SourceWorker thread has a new image but all threads in the PipelineWorker pool are busy the image gets discarded to prevent too much latency between creation and processing of an image.

5.2.1 Evaluation

To evaluate a possible performance gain of our multi-threaded architecture on systems with more than one processor core, we implemented a traditional single threaded image processing path alongside the multi-threaded one. Using these two paths, we ran some initial tests measuring the frame rate. We used a single system equipped with a quad core processor (Intel Core 2 Quad Q9400 @ 2.66 GHz) and deactivated three and two of the cores to simulate a single and dual core system respectively.

We used a CPU intensive filter chain for our measurements determining the possible gain under heavy load. We measured the amount of frames processed in one minute and ran this test 5 times for each configuration. The average results are depicted in Figure 14.

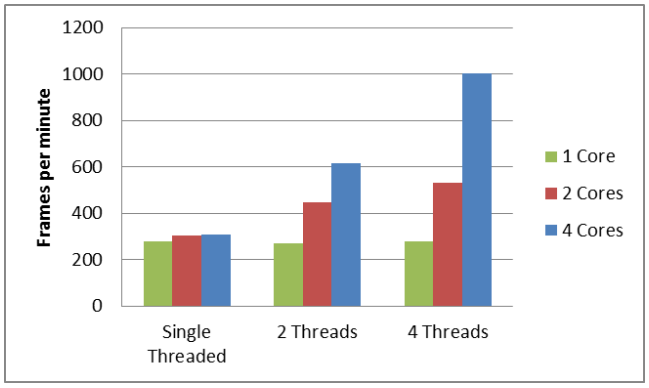


Figure 14: Results from our initial performance test.

Using the single-threaded approach frame rate is nearly identical for all processor configurations. The small performance advantage of the multicore configurations can be attributed to the fact that the GUI thread of the application as well as other processes running in the background can be executed on a separate core. Using two and four threads the performance on multicore configurations increases significantly. On the quad core configuration about 3.3 times more frames could be processed when using 4 threads (1002 fpm) in comparison to the single threaded approach (307 fpm).

6. CONCLUSION AND FUTURE WORK

In this paper, we presented LightTracker – a highly modular open source touch tracking toolkit. We listed the shortcomings of other open source touch tracking libraries we have discovered while building multi-touch setups and described how LightTracker tries to avoid these. The highly modular filter chain at the heart of our tracking solution and our improvements to the calibration process of multi-touch setups using camera lenses with non-linear characteristics have been presented. As an example of how to successfully adapt LightTracker to non-standard hardware setups, we demonstrated our gaming couch table.

The implementation part introduced the design of the plugin system and illustrated the creation of custom plugins as well as showing the multi-threaded approach of LightTracker and the performance gains which can be achieved with this architecture on multicore systems.

At the moment additional LightTracker plugins are still actively developed. Work on a multi-camera source plugin, stitching together the images of multiple cameras, has been started. LightTracker is being released for the Microsoft Windows platform under the GNU General Public License [10]. As all of the libraries used are cross-platform, future releases for other platforms should be possible.

7. REFERENCES

[1] Bederson, B. B., Grosjean, J., and Meyer, J. 2004. Toolkit Design for Interactive Structured Graphics. IEEE Trans. Softw. Eng. 30, 8 (Aug. 2004), 535-546.

[2] Bakker, S., Vorstenbosch, D., van den Hoven, E., Hollemans, G., and Bergman, T. 2007. Weathergods: tangible interaction in a digital tabletop game. In *Proceedings of the 1st international Conference on Tangible and Embedded interaction* (Baton Rouge, Louisiana, February 15 - 17, 2007). TEI '07. ACM, New York, NY, 151-152.

[3] Barakonyi, I., Weilguny, M., Psik, T., and Schmalstieg, D. 2005. MonkeyBridge: autonomous agents in augmented reality games. In *Proceedings of the 2005 ACM SIGCHI international Conference on Advances in Computer Entertainment Technology* (Valencia, Spain, June 15 - 17, 2005). ACE '05, vol. 265. ACM, New York, NY, 172-175.

[4] BBTouch. Available at http://benbritten.com/software/bbtouch-quick-start/.

[5] Community Core Vision (CCV). Available at http://ccv.nuigroup.com/.

[6] Dietz, P. and Leigh, D. 2001. DiamondTouch: a multi-user touch technology. In Proceedings of the 14th Annual ACM Symposium on User interface Software and Technology (Orlando, Florida, November 11 - 14, 2001). UIST '01. ACM, New York, NY, 219-226.

[7] Echtler, F. and Klinker, G. 2008. A multi-touch software architecture. In Proceedings of the 5th Nordic Conference on Human-Computer interaction: Building Bridges (Lund, Sweden, October 20 - 22, 2008). NordiCHI '08, vol. 358. ACM, New York, NY, 463-466.

[8] Esenther, A., Wittenburg K., Multi-user multi-touch games on DiamondTouch with the DTFlash toolkit *Intelligent Technologies for Interactive Entertainment*. Springer. 2005. pp 315-319

[9] Geller, T. Interactive Tabletop Exhibits in Museums and Galleries, *IEEE Computer Graphics and Applications* 2006 (Vol. 26, No. 5), pp. 6-11.

[10] GNU General Public License. Available at http://www.gnu.org/licenses/gpl.html

[11] Han, J. Y. 2005. Low-cost multi-touch sensing through frustrated total internal reflection. In Proceedings of the 18th Annual ACM Symposium on User interface Software and Technology (Seattle, WA, USA, October 23 - 26, 2005). UIST '05. ACM, New York, NY, 115-118.

[12] Hansen, T. E., Hourcade, J. P., Virbel, M., Patali, S., and Serra, T. 2009. PyMT: a post-WIMP multi-touch user interface toolkit. In Proceedings of the ACM international Conference on interactive Tabletops and Surfaces (Banff, Alberta, Canada, November 23 - 25, 2009). ITS '09. ACM, New York, NY, 17-24.

[13] Kaltenbrunner, M. 2009. reacTIVision and TUIO: a tangible tabletop toolkit. In Proceedings of the ACM international Conference on interactive Tabletops and Surfaces (Banff, Alberta, Canada, November 23 - 25, 2009). ITS '09. ACM, New York, NY, 9-16.

[14] Kaltenbrunner, M. and Bencina, R. 2007. reacTIVision: a computer-vision framework for table-based tangible interaction. In Proceedings of the 1st international Conference on Tangible and Embedded interaction (Baton Rouge, Louisiana, February 15 - 17, 2007). TEI '07. ACM, New York, NY, 69-74.

[15] König, W. A., Rädle, R., and Reiterer, H. 2009. Squidy: a zoomable design environment for natural user interfaces. In Proceedings of the 27th international Conference Extended Abstracts on Human Factors in Computing Systems (Boston, MA, USA, April 04 - 09, 2009). CHI '09. ACM, New York, NY, 4561-4566.

[16] Leitner, J., Haller, M., Yun, K., Woo, W., Sugimoto, M., and Inami, M. 2008. IncreTable, a mixed reality tabletop game experience. In Proceedings of the 2008 international Conference on Advances in Computer Entertainment Technology (Yokohama, Japan, December 03 - 05, 2008). ACE '08, vol. 352. ACM, New York, NY, 9-16.

[17] Leitner, J., Köffel, C., and Haller, M. 2009. Bridging the gap between real and virtual objects for tabletop games. Int. J. Virtual Reality 7, 4, 33-40.

[18] Magerkurth, C., Memisoglu, M., Engelke, T., and Streitz, N. 2004. Towards the next generation of tabletop gaming experiences. In *Proceedings of Graphics interface 2004* (London, Ontario, Canada, May 17 - 19, 2004). ACM International Conference Proceeding Series, vol. 62. Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, 73-80.

[19] Matsushita, N. and Rekimoto, J. 1997. HoloWall: designing a finger, hand, body, and object sensitive wall. In Proceedings of the 10th Annual ACM Symposium on User interface Software and Technology (Banff, Alberta, Canada, October 14 - 17, 1997). UIST '97. ACM, New York, NY, 209-210.

[20] Microsoft Surface. Available at http://www.microsoft.com/surface.

[21] Movid. Available at http://movid.org/.

[22] Multi-touch for Java. Available at http://www.mt4j.org/mediawiki/index.php/Main_Page.

[23] OpenCV. Available at http://sourceforge.net/projects/opencvlibrary/

[24] Peltonen, P., et al. "It's Mine, Don't Touch!": Interac- tions at a Large Multi-Touch Display in a City Centre. Proc. of CHI 2008, ACM. pp. 1285-1294.

[25] Qt, Cross-platform application and UI framework. Available at http://qt.nokia.com/

[26] Shen, C., Vernier, F. D., Forlines, C., and Ringel, M. 2004. DiamondSpin: an extensible toolkit for around-the-table interaction. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (Vienna, Austria, April 24 - 29, 2004). CHI '04. ACM, New York, NY, 167-174.

[27] Touchè. Available at http://gkaindl.com/software/touché.

[28] Touchlib. Available at http://www.nuigroup.com/touchlib/.