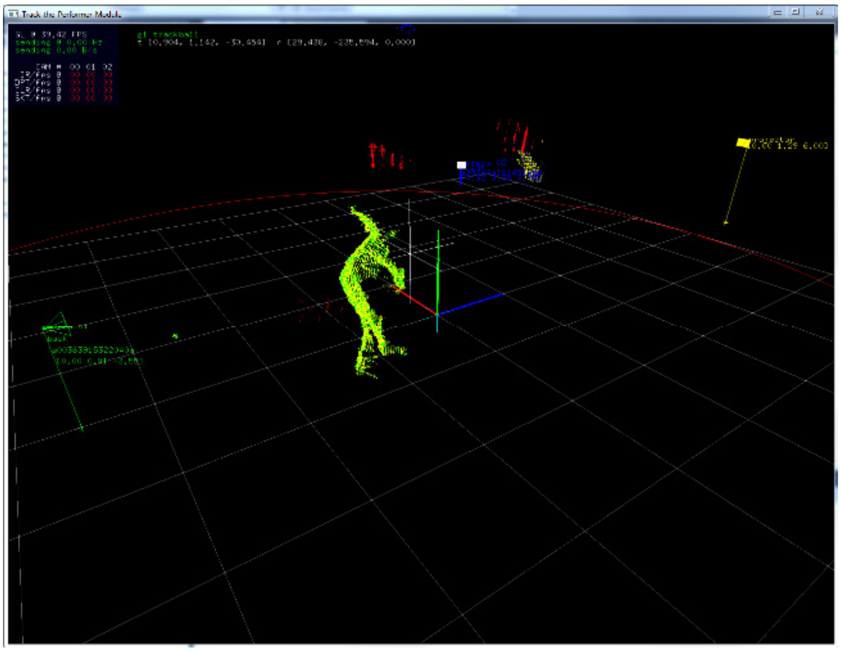
HydraNI

This project contains the image/data processing code, mostly used to realize the multi-Kinect tracking part of Aakash Odedra's most recent performance "Murmur".

A project Video can be found on: https://www.youtube.com/watch?v=fa9BvFgvk3k

HydraNI consists of two separate applications: One Body (acting as the Server) and an arbitrary number of Heads (acting as Clients). In order to provide a scalable multi-Camera setup, the Clients may be distributed on multiple machines, performing pre-processing of the captured depth data and send the result to the server via TCP (default port is 5556), which ultimately merges the tracking data and does the final processing.



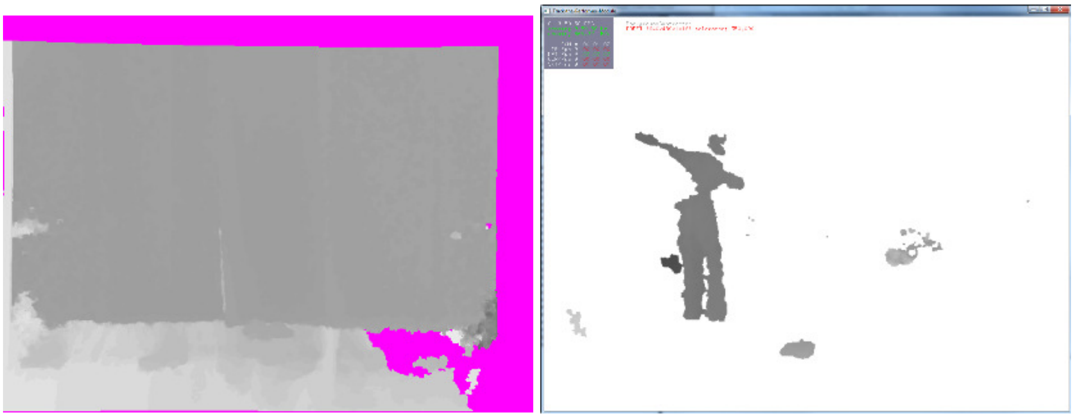
Heads by default open the first OpenNI capable device found on the system and start capturing, processing, and sending to the server socket. In the case of multiple devices connected to one single machine, serial numbers can be specified in order to assign a source device to a client application. Since we experienced some issues with drivers, especially when more than 3 cameras were connected to a single machine, there is a feature of auto-recovery, which re-initializes all the cameras whenever a single camera seems to be lost. Lost devices are detected by paying attention to the delay between frames – e.g. if a camera doesn't send for a second, it is considered lost and the camera context, including all cameras, is re-initialized. Since initialization takes a moment, it will be visible as a short freeze of the tracking and all of its outputs, which might be undesirable, depending on the application or its state, so user caution is advised when using this feature.

Pre-processing on HydraNI Head

Client-side preprocessing consists of five steps:

Background subtraction

Background models can be recorded and saved to disk (specified by camera serial) as 16 bit TIFF. The saved background model is loaded at application startup for each camera. If there is no background model available, no processing is performed at this step. The model itself is built using a slightly specialized running-average method of the current depth frames in order to handle depth noise and out-of-range areas. Whenever the model is recorded (and later on saved to disk), a snapshot of this running-average model is taken and from there on used for background subtraction.

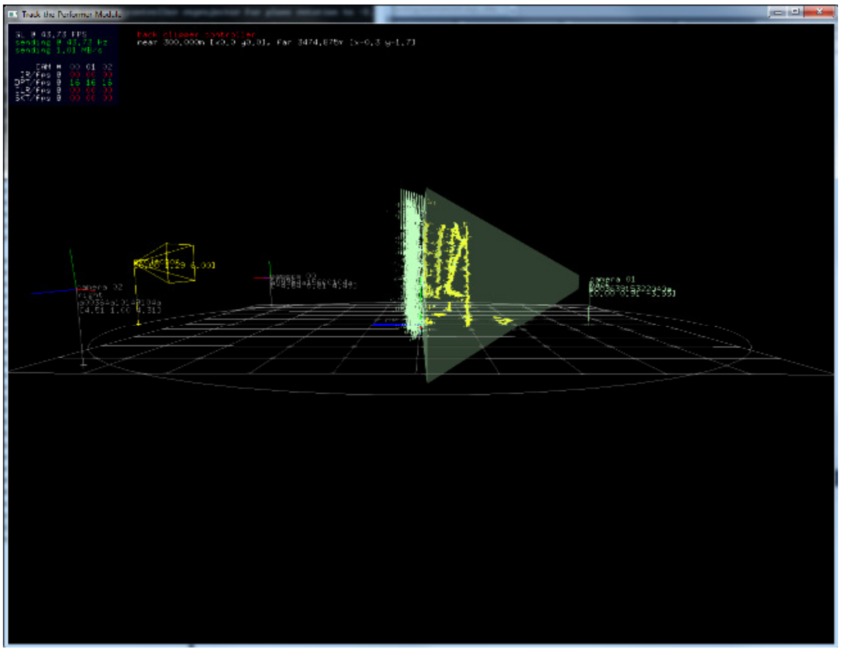


Morphologic closing

As structured light depth sensing devices notoriously suffer from high frequency noise, especially around object edges, a morphologic closing filter is applied to fill small holes, smoothen edges, and thus reduce movement along blob edges.

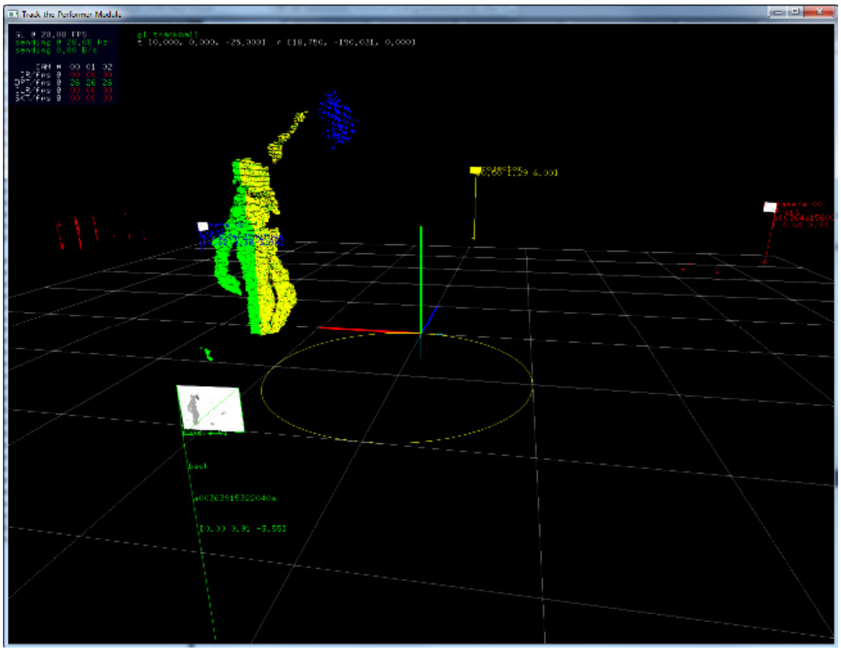
Frustum clipping

As an additional means to get rid of undesired objects and activity, it is possible to specify a clipping cone for each of the cameras individually. Especially tall distant objects generate more noise and thus contribute to the global error which seriously affects the calculation of the COM (center of mass), pulling it towards the object.



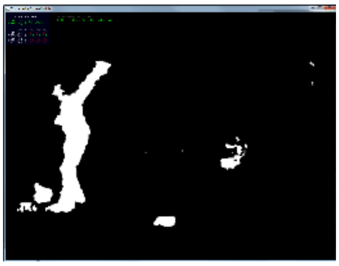
Cylindrical clipping

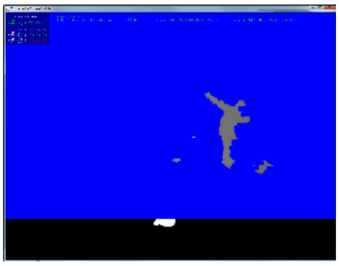
Cylindrical clipping is another way of getting rid of undesired objects and activity on or off stage, it was introduced since the stage for Murmur was circular and everything beyond the circle of fans can be considered not to be of interest. Cylindrical clipping simply disposes all points which are outside a circle in top-down view (thus outside of a cylindrical volume of infinite height), center and radius of which war configurable.



Perspective re-projection

Since the application was to do Augmented-Reality-style projections on stage, based on data captured by multiple, spatially distributed depth cameras, the remaining depth pixels are re-projected onto a hypothetic projector frame buffer, suitable for the visuals renderer. The resulting frame buffer represents the captured (and remaining, after processing) depth pixels, as they would be visible from the projector's point of view, given its intrinsic and extrinsic parameters. This is performed by calculating the real-world positions of each of the points represented in the depth frame, and projecting them onto the projector's view plane. Next to the projector's intrinsic and extrinsic parameters, this also requires those of each of the camera. Cameras' extrinsic and intrinsic parameters are stored on disk, in OpenCV matrix format, along the background model. File extensions are *.ext and *.extInv, *.int and *.intInv for extrinsic and intrinsic parameters and their inverse, respectively, and *.dist for camera lens' distortion coefficients.





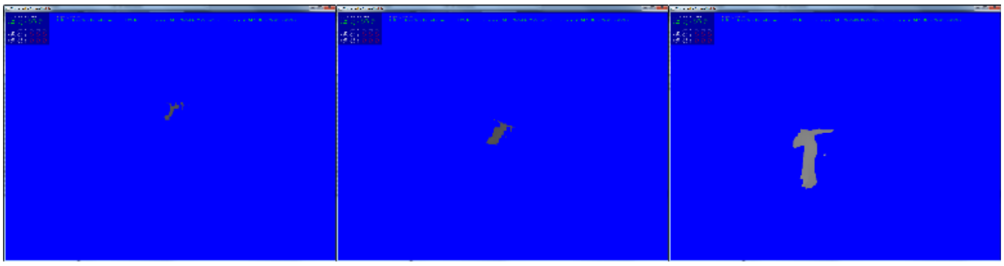
In case network bandwidth is an issue, and since the resulting pre-processed frames are highly suitable for compression due to their usually high amount of black pixels, a possible performance improvement would to compress the frames with zlib or any other realtime capable compression library.

Processing on HydraNI Body

The HydraNI server application gathers all incoming projector-point-of-view depth frames, and merges them for blob tracking and COM calculation.

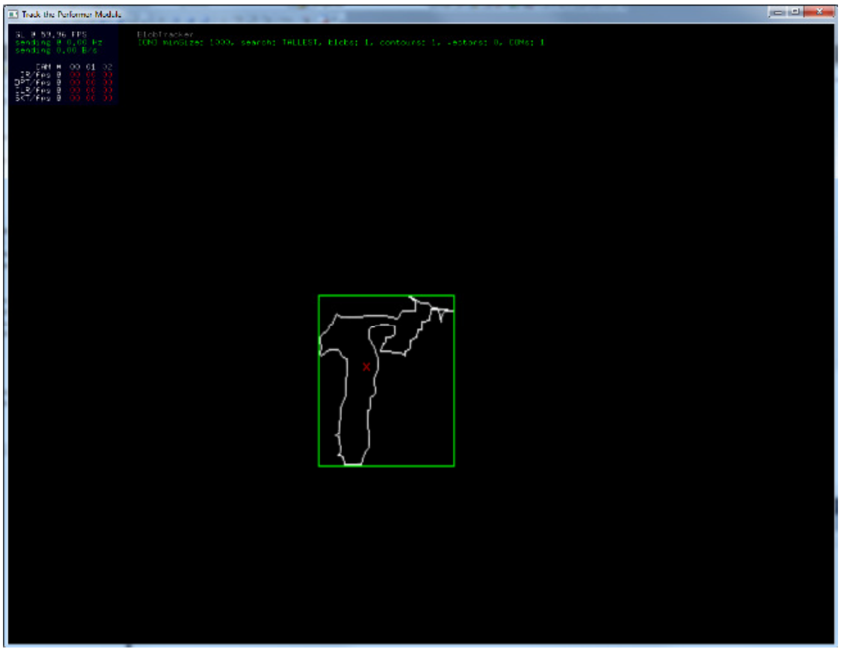
Frame Merging

All incoming frames, which already are processed to reflect projector-point-of-view, are merged in a common z-Buffer manner, where closer pixels overwrite distant ones. The resulting image represents the whole scene, as captured by all the cameras on stage, as it would be visible from the projector's position and with its field of view.



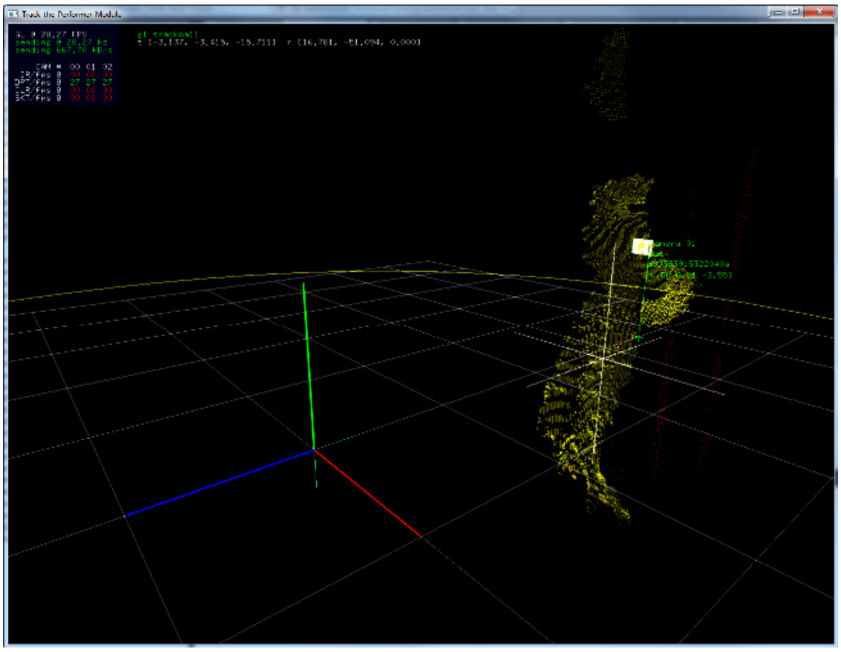
Blob detection and silhouette generation

On the merged image, a threshold operation is performed, followed by a connected component analysis (inner contours are ignored). Areas below a specifiable size limit are discarded. Either just the tallest contour or all contours are selected for sending to the renderer (sending of all contours is currently not fully implemented, though).



Calculation of COM

For delivering the COM (center of mass) to the renderer, the center of mass is calculated from all the 3D-points remaining from all points remaining after pre-processing and frame merging. A more accurate COM could be found by doing this pre-merging, since several occluded values are lost during the operation, on the other hand, overlapping regions would add several times. During experiments we found our current implementation to yield best results for our requirements. The current implementation discards the y-value and thus represents the COM position on the XZ-plane, but the code can easily be extended to deliver all three dimensions.



Datatype description

HydraNI Body writes and publishes (default port 5557) the data type BlobFrame, defined in the according HydraNI Bundle. See bundle documentation for a detailed type description.

Dependencies

HydraNI requires the HydraNIBundle to be present in the bundle directory. For building the code, additional libraries are needed. For using the MSVS projects, the according environment variables have to be set as follows. The versions, when stated, are the ones the application was built against. The networking part uses _2RealNetwork, which is a wrapper of the ZeroMQ networking library, tailored for the _2RealFramework.

Library, Version Nr.	**Environment variable name(s)**	**Environment variable value(s)**
_2RealFramework	_2REAL_FRAMEWORK_DIR	Absolute path to “kernel” directory
_2RealNetwork	_2REAL_NETWORK_DIR	Absolute path to “network” directory, containing folders “bin” and “lib”
OpenNI 1.5.7.10	OPEN_NI_INCLUDE OPEN_NI_LIB	Both usually set by installer
OpenCV 2.4.9	OPEN_CV_249	Absolute path to OpenCV API root directory, e.g. containing directory “include”
MongoDB C++ Drivers 1.0.0-rc1	MONGODB_DIR	Absolute path to MongoDB API root directory, e.g. containing directories “include” and “lib”
freeglut 2.8.0 (alternatively, glut also works)	GLUT_DIR	Absolute path to glut root directory, e.g. containing directories “include” and “lib”
glew 1.7.0	GLEW_DIR	Absolute path to glew root directory, e.g. containing directories “include”, “bin”, and “lib”
glm 0.9.3.1	GLM_DIR	Absolute path to glm root directory, e.g. containing include directory “glm”
ZeroMQ 4.0.4	ZMQ_DIR	Absolute path to ZeroMQ root directory, e.g. containing directories “include”, “bin”, and “lib”
boost 1.55.0	BOOST_DIR	Absolute path to boost root directory, e.g. containing include directory “boost”
libusb	LIBUSB_DIR	Absolute path to libusb root directory, e.g. containing include directory “libusb”

Supported platforms

The codebase of HydraNI is basically built and tested for Windows and Linux, the final version was only tested on Windows, though (Microsoft Visual Studio 2013). Any minor compiler errors should however be easily fixed when switching to gcc.