The Gestures of Flowing

Using PureData as a Backbone for Interactive Sculpture Animation,Video and Sound

Dr. Andreas Mahling
University of Music and Performing Arts Stuttgart
Urbanstraße 25, 70182 Stuttgart
Germany
email: andreas_mahling@hotmail.com

Abstract

The *Gestures of Flowing* is an installation which exhibits animated sculptures driven by sensoric input taken from an audience. Video as well as audio is considered as input and output and is processed in realtime by a PureData program patch. Furthermore sensor data is used to control hardware like motors, lamps, pumps and ventiles via a memory programmable control unit (SPS). The way how sensor input influences video, audio and control output is specified in scenario patches, out of which three will be described in detail in this article. To verify correctness of interaction scenarios even if sculpture hardware is not available, a small simulator will be introduced.

Introduction

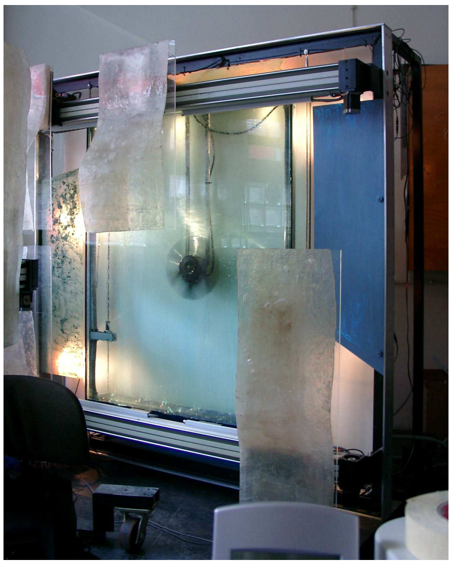
This paper describes a project, which is about developing an installation for controlling the change of a buildings facades appearance due to changes in its sourrounding interactively. The property of facades to change appearance over time in general is nothing new: depending on the shaping and material used, facades appear different when sun is shining or when it's raining. They cast shadows which change over time if light source is moving, light is reflected differently by the walls depending on whether they are wet or dry and they may even emit different sound depending on the strength of wind. Architects are aware of this aspect of building design since centuries: projecting figures on facades of baroque churches for instance were, among other objectives, consciously created under the aspect of how they would cast shadows and not just for to show themselves.

Thinking about this quality of facades as being some kind of interactiveness, our projects target is to raise the level of interactivity of facades by introducing sensoric/input and actoric/output elements and to control them actively. Sensors have the task to watch what is happening near specific locations of the facade and actors do react on that input, e.g. if a person bends down itself in front of a facade, this motion might be detected and tracked by a live-video input module and the data produced by this module might be used to have surfaces on the facade follow this movement by means of a spinmotor. Among the list of sensors are microphones and cameras but also some non-obvious ones like a "virtual ear" listening to continuously updated wheather data accessed through an internet link. Stepmotors for moving and turning surfaces, TV- and audio monitors for visual and aural feedback, coloured lights, pumps, ventiles and rotors belong to the list of actors and the striving possibility to actively play with different aggregate states of substances (gaseous, liquid, solid) as a part of an animated facade extends the scope of influence on its appearance even further; e.g. remember how hoarfrost may change a surfaces look.

Project Overview

The project *Gesture of Flowing* is a cooperation between the faculties of architecture and informatics at the university and the music conservatory at Stuttgart. It was launched and is lead by the *Institut für Darstellen und Gestalten 2 (IDG2)*, a division of the faculty of architecture headed by Prof. Herbert Traub. The *Department of Image Understanding*, headed by Prof. Paul Levy, a division of the *Institute of Parallel and Distributed High Performance Systems (IPVR)*, adds expertise and development resources in the field of picture processing and recognition and is responsible for what kind of control-data is extracted from the live video sources. This control data can be used for instance to detect movements or even stereotypes of movements, i.e. human gestures. The music conservatory originally was asked to take over all audio-related processing, e.g. tracking, manipulation and output of sound and to support the selection process of audio hard- and software. This responsibility has been extended to develop and maintain the central software platform used for receiving sensor input, for manipulating and computing control data and to control actors through this control data.

The IDG2 is the initiator of the project and responsible for overall project organization, the projects underlying ideas and the creation of animated facade prototype modules. Although a couple of different types of animated facade modules have already been manufactured it is sufficient for this article to concentrate on three instances of a single facade prototype to clarify the projects basic ideas, approaches and their realization. These animated facade prototypes are organized into two sideframes and one central frame. Each frame has shaped couloured surfaces which can be moved horizontally through stepmotors or turned up and down via spinmotors, coloured lamps, pumps, ventiles, a camera, piezo and dynamic microphones. The central frame also incorporates a rotor which may spread/dissipate liquid inside a basin conveyed by pumps. Fig. 1 shows the centerframe and a 3D illustration of a three-frame installation.



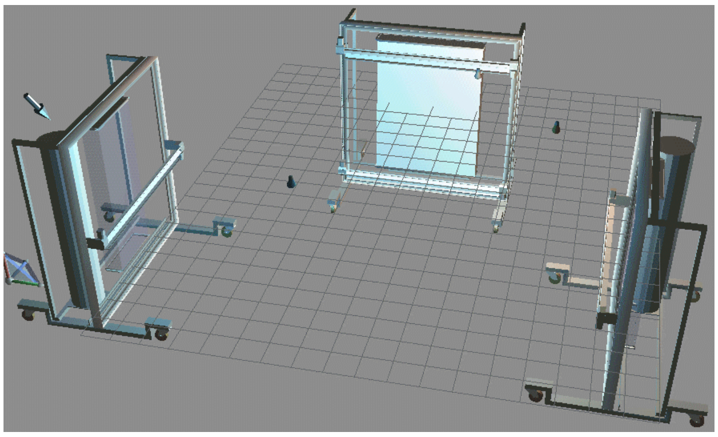
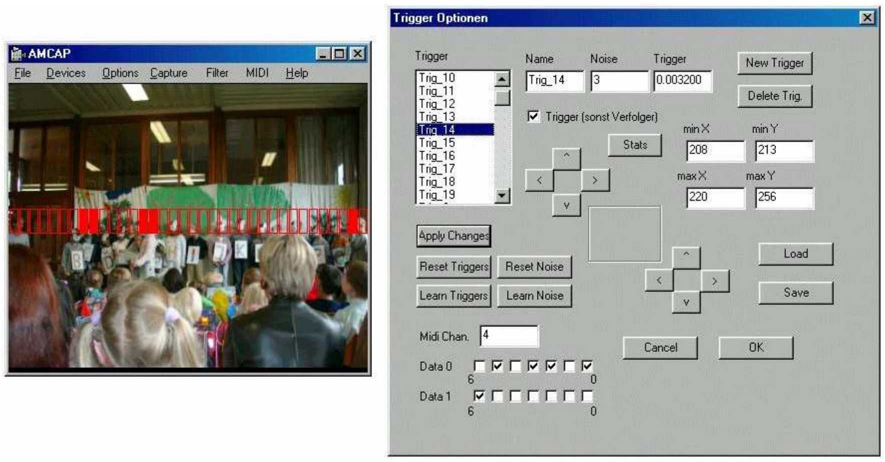


Fig.1: Photo of centerframe and 3D illustration of three-frame installation.

So far the three frames constitute an installation where the central frame is placed opposite to and sideframes are placed to the left and to the right of a potential visitor. This arrangement is used to make the visitor feel like being an integral part of the installation and to facilitate 3D motion tracking. At a later time prototypes should migrate into a twisted quader entirely made of glass which itself should become part of a facade.

All actors except those for video and sound input/output are controlled through a modular PC based SPS. Currently the project works with two interface cards providing control channels for 16 motors and a multitude of digital and analog outputs for controlling lamps, ventiles, pumps and rotors. On the PC side a dynamic link library (DLL) running under Windows 2000 provides access to the functionality of the SPS. This library is interfaced through a proprietary wrapper-DLL to a real-time multimedia processing software (PureData) responsible for taking sensor input, realtime data manipulation and generating control output.

Video input is handled via proprietary software. There have been several versions of this software each focussing on a different approach of how motion-data can be extracted from the live-video input in realtime. The current version, at a glance, allows for placing sensitive rectangular regions on top of the video window which get triggered if pixel-data underneath the trigger-region exceeds a defined amount of change(s), e.g. if all pixel contained by the trigger-region change their color from grey to red. Trigger-regions may have arbitrary size and position and the number of triggers to be used is determined by the user. Trigger constellations can be saved to and reload from disk and because data is stored in ASCII format one might even generate trigger clusters like spirals by external programs. The program also provides many features for tuning the program to properties of the environment "looked at", e.g. whether there is more or less light or adjusting itself to a specific color distribution prevailing the live video input. Due to higher demands with respect to computational ressources the video preprocessing software is running on three separate PCs, one for each frame, communicating their trigger data via MIDI to the main PC (containing the SPS and running PureData). Fig. 2 shows an example trigger setup and its settings dialog.



Trigger: Trig_10, Trig_11, Trig_12, Trig_13, Trig_14, Trig_15, Trig_16, Trig_17, Trig_18, Trig_19

Name: Trig_14
Noise: 3
Trigger: 0.003200

☑ Trigger (sonst Verfolger)

min X: 208
min Y: 213
max X: 220
max Y: 256

Midi Chan. 4

Data 0 ☐ ☑ ☐ ☑ ☑ ☐ ☑ (6 … 0)

Data 1 ☑ ☐ ☐ ☐ ☐ ☐ ☐ (6 … 0)

Fig.2: Viewport window showing an example trigger setup and its trigger settings dialog.

Audio and MIDI is interfaced through a Midiman Delta 1010 breakout box which offers 8 channels of electrically balanced analog inputs and outputs. The current setup uses a dynamic microphone for each frame, two audio monitors for the center frame and a single audio monitor per sideframe, thus leaving 4 free audio channels for future extensions. Additionally, contact microphones will be used to provide for some immediate audio feedback either caused by loudspeakers or stepmotor sounds. On the software side, PureData provides all what the project needs for processing audio and MIDI.

Backbone Architecture and Design Principles

The first team meetings in 2002, which also constituted the first contact of the music conservatory with the project, raised a couple of constraints typical to projects in general and particularly specific to projects in arts and music: few time and resources to realize what already had come in mind and the impossibility to specify in detail what should exist finally because it could not be known in advance whether scenario drafts would actually function in reality. Thus, the project operating environment would have to remain as flexible as possible especially because it was intended to gradually increase the number of scenarios at a later time. The lack of enough manpower also urged the use of standards and open software whenever possible.

The decision to use PureData was initially ruled by the request for processing audio, because programming environment resembles an interpreter, e.g. no compile-link-run cycle, because complete program sources were available, because program could be operated in a distributed manner on a PC cluster if additional computing resources would be needed in the future, because it was running on multiple platforms and because it represented a kind of standard which already had proven to run stable for more than ten years. Later, the idea came up to also control the SPS through PureData by interfacing the dynamic link library (DLL) of the SPS with a wrapper DLL meeting the requirements for PureData externals and to communicate preprocessed video data via MIDI, which significantly reduced the amount of proprietary programming. At that stage, PureData definitely became a backbone for the entire project because it was planed to be used for receiving all relevant sensor input - video cameras, microphones and incoming SPS messages - processing it and, except for video monitors, to control all actors mentioned in the introduction of this article. The availability of an OpenGL external for visualizing simulations of system behavior in a virtual 3D space extended the usefullness of PureData even further. On the other side, most application programming would have to be done in PureData, requiring know-how which was not available to project members other than those from the music conservatory. This came out to be a bottleneck which is currently solved by employing students of the music conservatory.

Many of the actors used, e.g. the stepmotors, provide a multitude of commands for moving and state requests, e.g. to set position, speed, acceleration to target speed or to ask for a motors current position. Commands are sent by writing values into specific registers. To set the movement speed of a stepmotor to 500, for instance, 500 will be written to register 1x103, where x=2..9, specifies the motor whose speed is to be set and 103 is the command to set speed. To set a motors target position a value has to be written into register 1x102, i.e. 102 represents the set-target-position command.

An early version for controlling actors written in C listed all relevant registers in a commented table, e.g. it contained registers 11102, 12102, 13102,..,19102 for controlling the target positions of the corresponding stepmotors along with a value field for supplying new positions interactively. Besides the fact that it is impossible to realize complex scenarios with internal interrelationships between sensors and actors that way, this table represented an abstraction level insufficient for flexible control and interactive scenario experimentation. The current PureData implementation, therefore, invented a *device abstraction layer* in which each actor is represented by a separate patch (Fig.3). Each such patch encapsulates an actors functionality in a more descriptive way, including means for command parsing, value range normalization, initialization and handling of internal dependencies or peculiarities. Actors can be addressed symbolically which is easier to handle and leads to more independence from hardware implementation and system software settings, i.e. changes in the underlying association of actors with their registers will not influence any scenario communicating with the SPS via the device abstraction layer. Commands are sent to actors by using these symbolic addreses and can be in a mnemonic or simple numeric style, however mnemonic commands always use normalized parameter values whereas the numeric style supports absolute values only. Normalization of parameter values vastly facilitates exploiting new ideas, because absolute value ranges, which do not just vary between distinct parameters but also between same parameters of actors located on distinct frames, don't have to be in mind at any time. 1.0 is always the maximum, independent of whether it is used for setting a motors target position or its driving speed.

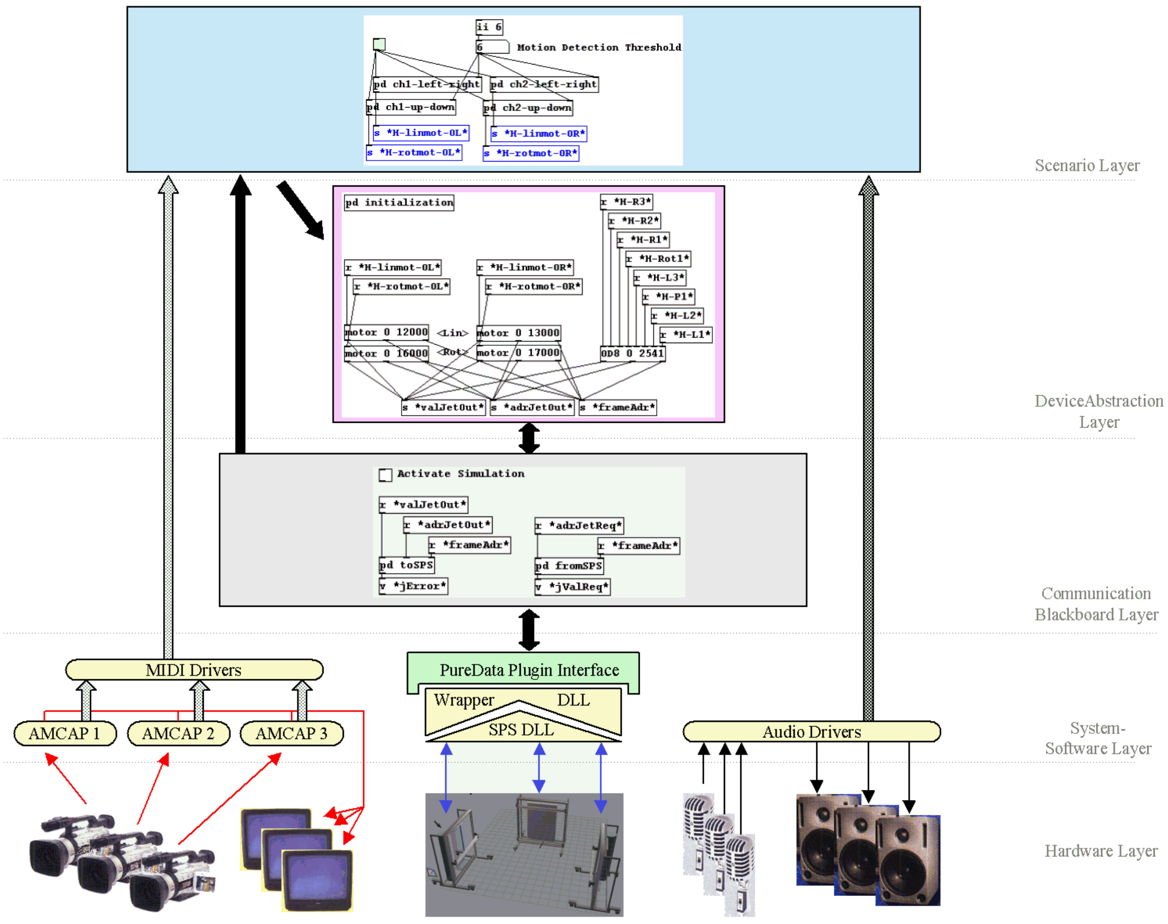


Fig.3: The backbone architecture.

Communication with the wrapper DLL is handled through one patch for sending commands and one for sending status requests. Both patches act like a blackboard, i.e. any control objects send their commands to a single instance of the patch making program space needed for communicating with the SPS independent from the number of actors and scenarios used. The wrapper DLL itself is instantiated twice, because two SPS cards have to be accessed.

To be able to verify correctness of scenario message flows even without having access to frame hardware, a patch for simulating hardware activity has been added. The simulation kernel, which is separated from the visualization of the simulation, is seamlessly integrated by providing the same communication interface as the program module used for accessing the SPS. Therefore a single switch is all that is needed to activate or deactivate the simulation.

Video input is currently received as MIDI note messages. That might seem strange but it was a straightforward approach, because it was easy to be implemented on the video preprocessing softwares side and was already well supported on PureDatas side, i.e. this approach saved a lot of time. Furthermore data resulting from the video preprocessing software had a structure that could easily be mapped on selected types of MIDI messages, e.g. information about the viewport quadrant from which a specific motion vector was computed was easy to be kept by the MIDI channel bits of MIDI channel messages. Last but not least MIDI was already well known by the developers of the video software.

Besides the features of realtime video processing offered by this software via DirectShow, two ways of extracting high level information from the video signal were choosen. The first one tries to detect points of motions in the camera viewport. Many such points of motion may exist at a time. Upon detection of a point of motion, a MIDI note on message is sent, encoding a two dimensional vector specifying strength and direction of the motion detected where big values of x and y coordinates mean a strong or fast motion. Color information associated with each motion vector is through the note-off part of the MIDI note message. The second approach makes use of triggers which can be arbitrarily placed within the cameras viewport and is already described indepth in the project overview of this article.

Scenarios

A scenario is a specific setup of how actors are controlled, usually influenced by sensor input and realized as a PureData patch. Generally, two categories of scenarios can be identified: those which are acting autonomuous and those which react on sensor input. Autonomuous scenarios are well suited to still let the environment act and be perceived interesting by the audience, if few or no significant input is received by the sensors or at least if no output is generated by the algorithms used to evaluate sensor input. Current scenarios have been built mainly to check proper working of interfaces, e.g. for video, audio and the SPS. Three of them will be described in the following sections of this chapter.

Direct Control

The *DirectControl* scenario is a means for direct manipulation of actors. It provides a window containing user interface widgets like sliders, knobs or toggles for indiviudal control of each actors parameters (Fig.4). For selected parameters common to a group of actors, like for instance the speed parameter of stepmotors, a global control is possible too. The *DirectControl* scenario serves very well if specific states of frame actoric have to be setup. In conjunction with the possibility to memorize such setups, actoric sequences can be designed and played back, which otherwise could not have been realized through a simple algorithmic automation. The playing of such sequences can be synchronized either to an internal clock or can be triggered by external events. Because PureData does not provide a preset storage mechanism like MAX so far, a proprietary solution was developed as a collection of PureData patches.

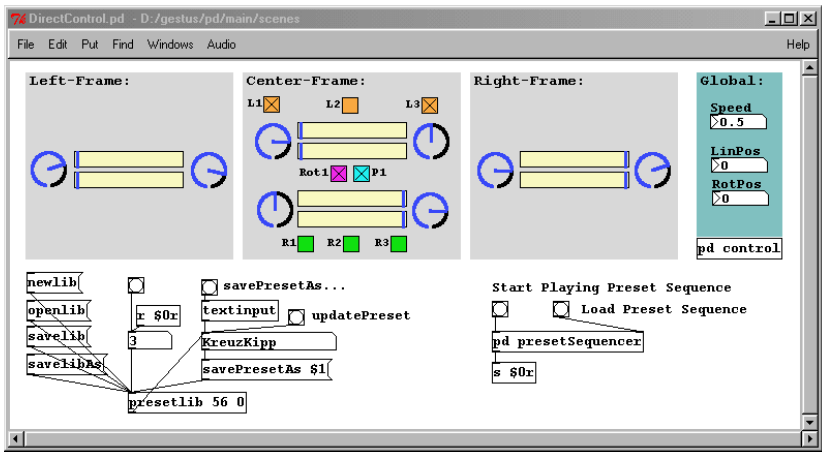


Fig. 4: The User Interface of the *DirectControl* Scenario.

For efficiency reasons, design and testing of scenarios, i.e. PureData patches, even without access to actors, was a project requirement originating from the amount of people working at different locations, the impossibility to make copies of the hardware in a cheap and easy way and last but not least because of the immobility of the frame environment itself. Therefore the *DirectControl* scenario is complemented by a simple simulation of actor activity, which can be used if real frame hardware is not present, providing visual feedback of each virtual frames current state through a monitor window. Currently only stepmotor movements at constant speed are simulated but even at this early development stage of the simulator its operation has facilitated the verification of message flows of PureData scenario patches enormously. An advanced version of the simulator, planned for the near future, will be based on GEM, an OpenGL external for PureData. It will cover more properties of the hardware for simulation and by using 3D representations of frames and actors, hardware independent demonstrations of the entire project or specific scenarios will be possible.

MIDIKeyToSpeed

MIDIKeyToSpeed is a scenario which uses MIDI data for controlling movement speed of stepmotors. It uses a proprietary PureData patch for playback of polyphonic MIDI sequences. Because PureData is unable to import MIDI Standard Files directly they are primarily converted to ASCII before playback relevant data is loaded. This data is stored in a numeric table to allow random access to sequence data for generating time synchronuous variations in realtime. Current scenario version uses pitch data for controlling the speed of transversal stepmotors of the centerframe, i.e. motor speed is proportional to a notes pitch. Because the centerframe of the installation incorporates four transversal stepmotors, compositions with up to four voices can be used. This can be increased to 8 voice polyphony if transversal motors of sideframes are considered too.

An interesting feature of this scenario is that stepmotors themselves make sound when spinning and that perceived pitch of this sound is related to the spinning speed. This leads to a playback of the MIDI data through stepmotors instead of loudspeakers which, due to nonlinearities inherent to this "reproduction system", introduces an interesting deviation from the original composition while still keeping significant resemblance with it at the same time.

TrackMotionInVideo

As mentioned in the introduction of this paper, one of the initial ideas for the entire project environment was, although rather idealistic, to let frames look at the audience, detect specific predefined stereotype movements or gestures of people and let frames react on that in a deterministic manner. To increase reliability of motion tracking a setup of three video-cameras, one on each frame, was used. Due to the positioning of the frames - one each to the front, left and right side - tracking of motion in all three directions is possible.

TrackMotionInVideo is a single frame scenario, i.e. it considers motion vectors based on the video signal of a single camera only. It is based on an early version of a proprietary video preprocessing software, refered to earlier in this article, which detects points of motion in the video signal and transmits the corresponding motion vectors via MIDI. Over a given timespan, lets say the accumulation interval, motion vector components are accumulated within the PureData patch for each direction separately, left, right, up, and down, starting with a value of zero. Then, if accumulated value exceeds a specific threshold, a stepmotor is told to move in that direction. To avoid intersections in responsibility of actors for responding on tracked motions, camera viewport is splitted into 4 quadrants: top-left, top-right, bottom-left and bottom right, each acting as a source for corresponding stepmotors of the centerframe, i.e. motion vectors stemming from analyzing video signal in the top-left quadrant control the top-left pair of stepmotors, one for traversal along the horizontal axes and one for spinning up and down. A collision control algorithm prevents traversal stepmotors from moving across boundaries, i.e. the end of the shaft, which already is a natural physical limit, on one side and the vertical boundary of the corresponding viewport quadrant on the other side. Rotating stepmotors cannot interfere with one another. By using at least two instances of *TrackMotionInVideo*, one per frame orthogonal to each other, a complete motion tracking in 3D space is possible.

A limitation of the current implementation of this scenario is that, if there is a predominating direction over a period of time, stepmotors will just move up to their boundaries and stop and are likely not to move back unless enough motion is detected in the opposite direction. This might be resolved by means of a reset which moves motors to an initial position, e.g. the mid of each viewport quadrant, if motors did not move over some time even though enough motion was detected.

Future Work

One of the main objectives of the project is to have the environment act interactively depending on visual and aural sensor input. Cooperation with the music conservatory was initially started to support selection of audio hard- and software to process audio input in realtime. Selecting PureData for audio processing however quickly lead to the point to also use it for controlling the SPS and processing (high-level) video input. This caused a significant increase in expenditure for patch design and implementation. Sampled audio, so far, has been neglected badly. Therefore an important next step will be to add scenarios. e.g. for alternating recording and playback of audio. Before playback, audio will be processed in a variety of ways, like reordering of record-sample slices, filtering etc.

Currently the project operates on two types of sensor input - video and audio - and three types of actor output - video, audio and the SPS. Interdependencies exist only between video input and SPS output and video in- to video output. This will be extended to also incorporate *audio input to SPS output* and *video input to audio output.*

The kernel for simulating hardware activity will be amended to also simulate lamps, ventiles, pumps and rotors. For the stepmotors, additional parameters like driving speed and acceleration will be included for a more realistic simulation. A three dimensional visualization for demonstrating simulated hardware activity will be developed and based on GEM, an OpenGL external for PureData. Last but not least, the number of available scenarios will be increased to allow more variation in environment activity and to improve its adaptability to distinct exhibition places.

References

[Levi2003]
Paul Levi, Director of the Department of Image Understanding, a division of the Institute of Parallel and Distributed High Performance Systems (IPVR): http://www.informatik.uni-stuttgart.de/ipvr/bv/bv_home.html

[Puckette2002]
Miller Puckette. PureData 0.36 Software Link: http://crca.ucsd.edu/~msp/software.html

[Traub2002]
Herbert Traub, Institut für Darstellen und Gestalten (IDG2): http://www.uni-stuttgart.de/idg2/fra_inst.html , *Gestures of Flowing* Project Link:http://www.kunst.uni-stuttgart.de/traub/gestus/html/frame.html

[Zicarelli2001]
David Zicarelli, MAX Reference Manual: Tutorial and Topics. Published by Cycling '74, 2001