INTERACTIVE MEDIA AND DESIGN EDITING FOR LIVE VISUALS APPLICATIONS

Pascal Müller
Computer Vision Lab, ETH Zürich

Simon Schubiger-Banz
Swisscom Innovations

Stefan Müller Arisona
Computer Systems Institute, ETH Zürich

Matthias Specht
Anthropological Institute, University of Zürich

Keywords: Interactive Content Creation, Authoring Systems, Design Computation, Interaction Techniques, Live Visuals.

Abstract: This paper describes novel concepts for the interactive composition of artistic real-time graphics, so-called *live visuals*. By establishing two fundamental techniques dealing with the structured media integration and the intrinsic design process, we significantly increase the efficiency of interactive editing in live visuals applications. First, we present a media manager that supports the user in both retrieval and utilization of automatically annotated digital media. The computer-assisted application of individual media items permits the interactive control of non-linear editing (NLE) of video in real-time. Second, we optimize the design process by introducing the *design tree*, which collects and organizes the artist's work in an intuitive way. Design tree operations provide interactive high-level editing methods which allow for exploration, combination, reuse, and evolution of designs before and particularly during the performance. We examined the effectiveness of our techniques on numerous long-lasting live performances from which representative examples are demonstrated.

1 INTRODUCTION

From the very beginning, interactive graphics systems have been used for creating art (Sutherland, 1963). They have fascinated and deeply influenced visual performance artists who would eventually replace analog video mixing and effect consoles in favor of computer systems. Among other movements, "VJing", referring to Video Jockeys (VJs) who compose live visuals at electronic dance music events, is recently becoming increasingly popular. An excerpt from a typical live visuals performance is illustrated in Figure 1. Performing such artistic real-time graphics live results in a tremendous need for state-of-the-art hardware and software systems. Today's systems virtually satisfy these needs in terms of performance and real-time processing. However, existing software tools mostly ignore the artistic design process before, and particularly during the performance. Furthermore, they don't support the user in dealing with large media libraries or in applying individual media items in an "intelligent" manner. Hence, this work presents novel approaches to the issues of media utilization and interactive designing in live visuals applications.

First, we present a media manager, which supports the live visuals artist in retrieval and utilization of digital media based on annotated metadata. Using well-known computer vision methods, we automatically augment video clips with metadata by reverse-engineering their original shot list. Besides enabling efficient media retrieval, the metadata permits the media manager to assist interactive non-linear editing (NLE) of video in real-time. A video clip's individual shots are restructured in a "non-linear" style by the live visuals system. For example, shots are rearranged in order to fit musical features extracted in real-time. Hence, Eisenstein's *vertical montage* theory on the articulation of film and soundtrack (Eisenstein, 1994) can be approached in a live context using annotated media files and real-time audio analysis.

The second contribution addresses the problem of

Müller P., Schubiger-Banz S., Müller Arisona S. and Specht M. (2006).
INTERACTIVE MEDIA AND DESIGN EDITING FOR LIVE VISUALS APPLICATIONS.
In *Proceedings of the First International Conference on Computer Graphics Theory and Applications*, pages 232-242
DOI: 10.5220/0001358502320242

Figure 1: Real-time graphics composed live by a performing artist. The snapshot sequence represents a 5 minute excerpt from a typical live performance consisting of abstract imagery, video loops, font layouts and a live video stream.

how artwork is dealt with during live performance. Rather than forcing the artist to fit into a fixed "preparation *vs.* performance" scheme, we provide mechanisms that give the artist freedom on which level individual design goals are placed. We introduce the high level concept of the *design tree*, which stores and organizes the artist's designs. The tree's nodes, representing individual designs, emerge by interactive composition or reuse during preparation, or as results of live composition during performance. Designs are "activated" (i.e., rendered) by selecting one or multiple design nodes. The series of activated designs results in a *design path* (Simon, 1996), representing a temporal plot of the actual performance. In order to allow for smooth transition between different designs, we provide a number of design tree operations, such as merging or mixing design nodes.

The paper starts by briefly discussing relevant related work. Section 3 provides an overview of a typical live visuals system and how the media manager and the design tree have been integrated. In Section 4 and Section 5 the main contributions listed above are presented in detail. Section 6 will show how the actual implementations of the media manager and the design tree are used to create effective live visuals. The paper concludes with final remarks in Section 7.

2 RELATED WORK

There is a wide range of existing software tools for creating live visuals. With the growing popularity of VJing, a large number of custom tools evolved (a comprehensive listing is found at (VJCentral, 2005)). Many of them resemble a digital counterpart of analog video mixers, where multiple video sources can be mixed and overlaid with visual effects. Others pick up the video mixing concept, but add features for digital compositing or rendering of arbitrary geometry, going far beyond the possibilities of analog video mixing (GarageCube, 2005). While most of these tools do a great job and are typically easy to use, their lack of generality imposes a major problem for visual artists that wish to go beyond predefined designs and without carrying a "footprint" of the software they were created with. Therefore, at the other end of the spectrum, applications with a general approach to creating live visuals (and often music as well) exist. They give a lot more freedom to the artist, but at the same time typically require at least some programming and signal processing knowledge. Two examples are Max, which has its origins in music and audio processing, and Touch, which roots in computer graphics modeling and animation.

Max (Pukette, 2002), and in particular Max/MSP and Jitter, represents a family of graphical programming environments. Max has become the *de facto* standard for many sound and visual artists. By applying the graphical programming paradigm objects of different types are interconnected to build *patches*. The patches represent a running program, and they as well serve as the user interface for interactive control. Therefore a programmer can create a "performing instrument", which can be used without programming skills. However, Max lacks means of organizing multiple patches beyond file system browsing or copy and paste. Thus, it is very hard creating several hours of visuals performance where a large number of differing designs is seamlessly arranged and mixed.

Derivative's *Touch* (its ancestor, *Houdini* was used to realize an interactive dance floor at the SIGGRAPH 98 Interactive Dance Club event (Ulyate and Bianciardi, 2002)) approaches the above problem by providing different environments for different levels of interaction: *Visual synthesizers* can be designed in Touch Designer, they can be mixed in Touch Mixer, or just played in Touch Player. At every level, the visual artist can interact (i.e., perform) in real-time.

In our work, the design tree navigation and manipulation methods act as the central means of design creation and modification. There is no distinction between preparation and performance, and our work emphasizes the actual "live design process" of a visual performance. This scheme not only addresses the mixing issues of long performances, it goes a step further and allows for true interpolation of different designs.

Another important task when compositing live visuals is the application of prepared media files. Although interactive retrieval and annotation systems have been available for some time (e.g., (Tseng et al., 2002)), to our knowledge there exists no live visuals tool with an integrated media manager that incorporates metadata such as MPEG-7 annotations (Manjunath et al., 2002), making our solution a novelty in this area.

Audio analysis has been used for controlling and aligning visual parameters in a scene graph (Wagner and Carroll, 2001), as well as together with video segmentation methods (Lienhard, 1999; Rui et al., 1999) for automatic or semi-automatic alignment of music and video in interactive offline systems (Foote et al., 2002) and in non-interactive real-time systems (Jehan et al., 2003).

3 SYSTEM OVERVIEW

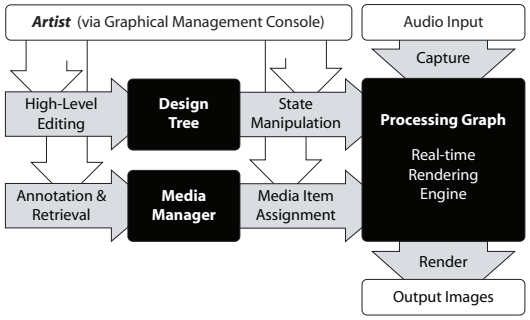

Figure 2: Overview of the SOUNDIUM system. Our two contributions, the design tree and the media manager (left), increase the efficiency of existing live visuals systems (right).

From a system architecture viewpoint, the common denominator of existing tools for live visuals is real-time rendering engine, which is controlled by a user interface. More specifically, the user interface can manipulate a *processing graph*, which is processed by the rendering engine. The produced output images are typically directed to a preview monitor and one or multiple video projections. In our work, this set-up is enhanced by a media manager and the design tree. The system illustrated in Figure 2 has been implemented in a proprietary live visuals system called SOUNDIUM. Implementation details haven been given in (Schubiger and Müller, 2003). For the remainder of this section we shall focus on concepts that are essential for understanding the functionality of the media manager and the design tree.

The processing graph is a directed graph of interconnected *processing nodes*, whereof each of them fulfills a certain purpose such as performing a 3D transformation, drawing a polygon, or calculating an audio signal level. Each processing node consists of a number of *input* and *output ports*, which are used for communication. The edges of the graph, interconnecting inputs and outputs, are called *connections*. With the processing graph, audio signal flow processing can be modeled as well as the hierarchical structures describing visual objects and their relations within a 3D world (similar to the well-known scene graph concept (Strauss and Carey, 1992)). Internally, the processing graph is stored as a sequence of SL2 statements. SL2 is an assembler-like scripting language designed for graph manipulation. Because of its simplicity, this textual representation is well suited for refactoring methods carried out by high-level design operations in Section 5.

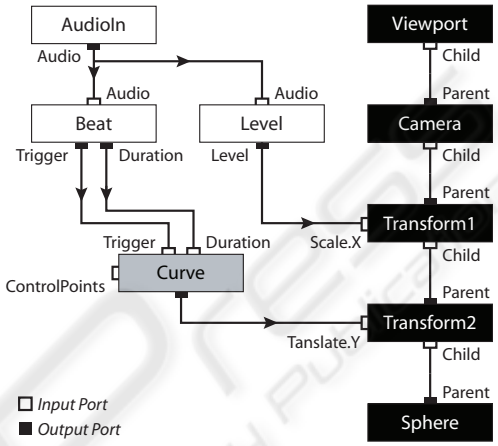

Figure 3: Simple processing graph example which represents an animated sphere driven by audio. White colored nodes depict the audio signal flow and black nodes represent the 3D scene graph. Additionally, an animation curve node is illustrated in grey.

A simple example is illustrated in Figure 3. The scene graph consists of a viewport, a camera, two transformations, and a node that draws a sphere. Simultaneously, the incoming audio is captured and processed, and the output of the level node is connected to the *Scale.X* input port of the first transformation node. The beat extrapolation node controls an animation curve node, effectively setting the duration of the curve to the beat duration, and restarting the curve with every beat trigger. Finally, the curve output is connected to the *Translate.Y* input of the second transformation. Typically, only a few of all available input ports are connected. An example is the curve nodes' *ControlPoints* input (all others have been omitted for simplicity). These ports are available for external value assignment by the user interface.

To align visual content to music features, *real-time audio analysis* is needed to extract the latter. In SOUNDIUM, audio analysis methods are realized in terms of processing nodes. Currently, the system provides methods for spectral analysis, beat and onset detection (Scheirer, 1998; Goto, 2001; Brossier et al., 2004), and part detection (e.g., refrains) based on similarity analysis of the signal's spectral composition (Foote et al., 2002). Typically, the audio analysis nodes deliver continuous parameters, such as spectral

levels or distributions, and discrete parameters, such as note onset triggers. How the parameters are connected to visual content is an artistic choice. However, SOUNDIUM supports the artist with a number of methods for the manipulation of audio analysis results in order to allow a better match to practical artistic goals, e.g., by providing dedicated processing nodes for modifying extracted rhythmical structures. Hence, manifold correlations between rhythm and visual content can be created.

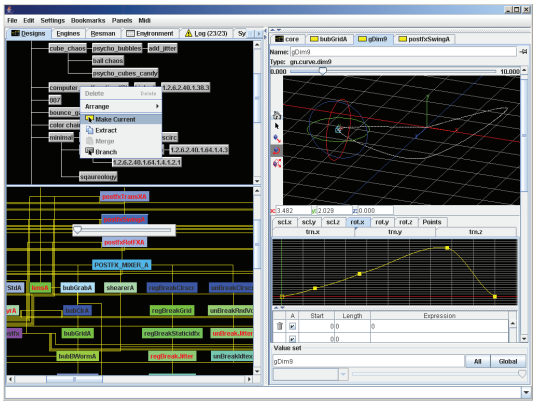

Figure 4: Screenshot of the graphical management console. Top left: Design tree which permits high-level editing. Bottom left: Processing graph representing the current system state. Right: Node inspector for modification of a selected processing node.

SOUNDIUM includes a *graphical management console* where interaction between the performer and the system takes place (see Figure 4). The main task of the management console is to maintain the current system state and consistent views of it. In addition, interactive state modification, high-level design manipulations, and media management takes also place through the console. The SOUNDIUM system state is represented as a highlighted node in the design tree (Figure 4, top left). This design node has an expanded view as a processing graph (Figure 4, bottom left). Not shown in Figure 4 are the design's textual representation (SL2) and its visual rendering. Modifications can simultaneously take place through a node inspector (see Figure 4, right) or connected MIDI controllers. Changes are reflected in all views immediately. The management console is non-modal, enabling direct access to every object and follows standard interface conventions as well as a minimalist approach for easier learning. Several accelerators and shortcuts ensure a high degree of interactivity for the experienced user. Because error states are fatal during a live performance, modifications are statically checked against boundary conditions and type safety (e.g. input and output port types) before being applied to the system state. Such errors will lead to an invalid design which is signaled to the user without interrupting the performance.

4 THE MEDIA MANAGER

A typical media library of a live visuals artist contains several gigabytes of pictures, video footage, geometric models, and so on. Hence, an efficient media manager is needed to allow for quick interactive access, which is crucial during live performance. In SOUNDIUM, the media manager extracts metadata in a "pre-production" phase by automatically scanning media files according to their type. In addition, the media manager supports manual annotation of media files (e.g., for semantic content descriptions which cannot be acquired automatically). The resulting metadata is stored in XML format based on the MPEG-7 description standard (Manjunath et al., 2002). The digital library with its incorporated metadata can be accessed by using fast and intuitive high-level retrieval mechanisms. The media manager's GUI is embedded in the graphical management console.

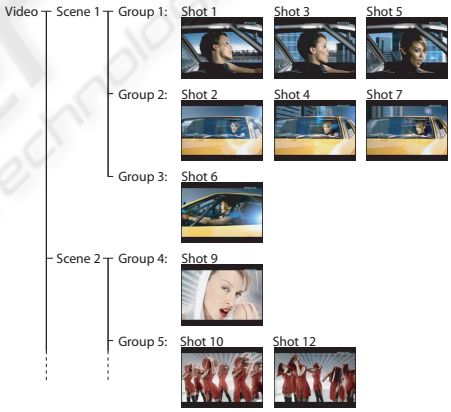

Figure 5: The hierarchical structure of a video clip reverse-engineered by the media manager (schematic view).

4.1 Video Clip Integration

Since most live visual performances include video footage, the video clip is the most important media type a visual artist is working with. Traditionally, footage is prepared manually using video editing tools, which is a very time consuming task. Our approach employs automatic methods, which considerably facilitate dealing with footage and reduce time consuming manual tasks to a minimum: The media manager analyzes the unedited footage using video segmentation techniques for shot boundary detection (Lienhard, 1999) and video abstracting techniques for

scene determination (Rui et al., 1999). The latter organizes the clip into *scenes*, *groups* and *shots*. Thus, a video clip is comprised of several scenes containing several groups, and a group itself consists of all visually similar shots within a scene (Figure 5). As a side effect of the scene determination algorithm, shots can be ordered within a group according to their group distance measure. This results in a measure for a group's "best" shot, which the visual artist will most likely use during performance. Furthermore, the media manager analyzes the motion of each shot, which results in a camera movement classification (pan/tilt/zoom/roll), and extracts the representative keyframes of each shot, which can be used for browsing. If desired, the user can modify the automatically generated video clip structure (e.g., by manually changing shot boundaries) and add content descriptions. For storage, the clip's source file is not modified, all editing information is stored exclusively in the metadata. Thus, the original clip is kept applicable to all kinds of scenarios.

For browsing video clips, the user has the choice of a temporal or a structural view. Particularly, the structural view gives an intuitive overview of the video clip. If the user selects a shot and its destination (i.e., a dedicated input port of a processing node), the media manager streams the corresponding frames to the rendering engine. In order to avoid long seek times within a video clip, each clip is encoded with keyframes at shot boundaries after the preparation phase. If requested, the engine caches the frames, which will remain available for random access later on.

4.2 Interactive Non-Linear Editing

Besides better retrieval capabilities, the extracted metadata of clips allows for a new form of video footage utilization: Interactive NLE of video, i.e., the (semi-)automatic rearrangement of a video clip's individual shots in real-time. In order to align live visuals to music, our approach applies film music theory in the reverse direction: The most popular film music procedure is to conduct the music according to given *visual action points* of a completely finished movie (Gorbmann, 1987). A visual action point is usually associated with a classical film cut, but it can also be within a continuous shot (e.g., the beginning of a pan) or refer to arbitrary types of dramatic events. In our case, visual action points have to be created in real-time for given "musical action points" resulting from audio analysis, for example extracted bar or beat borders may enforce cuts. Following these rules, the (short) clips of the dancers in Figure 1 have been synchronized to the incoming beat and extrapolated beat-durations by non-linearly stretching the clips between two beat boundaries.

In SOUNDIUM, the generation of visual action points is realized in terms of dedicated processing nodes for computer-assisted NLE. During performance, the user has interactive control over the selection of video footage and the node's configuration parameters. For instance, the user can assign a whole video scene or multiple groups to the NLE node, or tune editing parameters such as "cuts per second". The node then analyzes the associated metadata and – according to its configuration – decides which shots finally should be played, and how fast and for how long. SOUNDIUM includes NLE processing nodes implementing different editing techniques (Eisenstein, 1994) ranging from the functionality given above (simulating visual action points) to completely audio-independent editing.

5 THE DESIGN TREE

In our case, a *design* is a complete description of the processing graph, including its nodes, value vectors, and edges. On a more abstract level, a design directly reflects the realization of an artistic idea. The artist's designs are stored in the *design tree*, a hierarchical data structure, where each node contains information about how the processing graph is to be modified in order to realize a design. Changes to the system state (by using the graphical management console) result in modification of the processing graph and, if desired, also in new nodes in the design tree.

5.1 Realization

In its simplest form, the design tree can be seen as a multilevel undo/redo facility: All user actions manipulating the system state are recorded and can be undone. These state manipulations are recorded as SL2 statements representing individual processing graph changes. The user can decide to commit a design to the design tree, where a new design node is created. When a design is committed, the minimal sequence of SL2 statements yielding the system state is computed and called the *normal form* of a design node.

During the design process, several design nodes are committed by the user in sequence with each node representing a revision of a previous design (Figure 6-b/c). This is similar to a file versioning system (Cederqvist, 1993) that stores differences from one revision of a file to the next. Like in a versioning system, the user can go back to any previous design (Figure 6-d) and start a new branch (Figure 6-e), exploring a variant of a design. Thus, branching transforms the linear sequence of designs into a tree.

A natural ordering of nodes by time (revisions) takes place during the design process. However, this

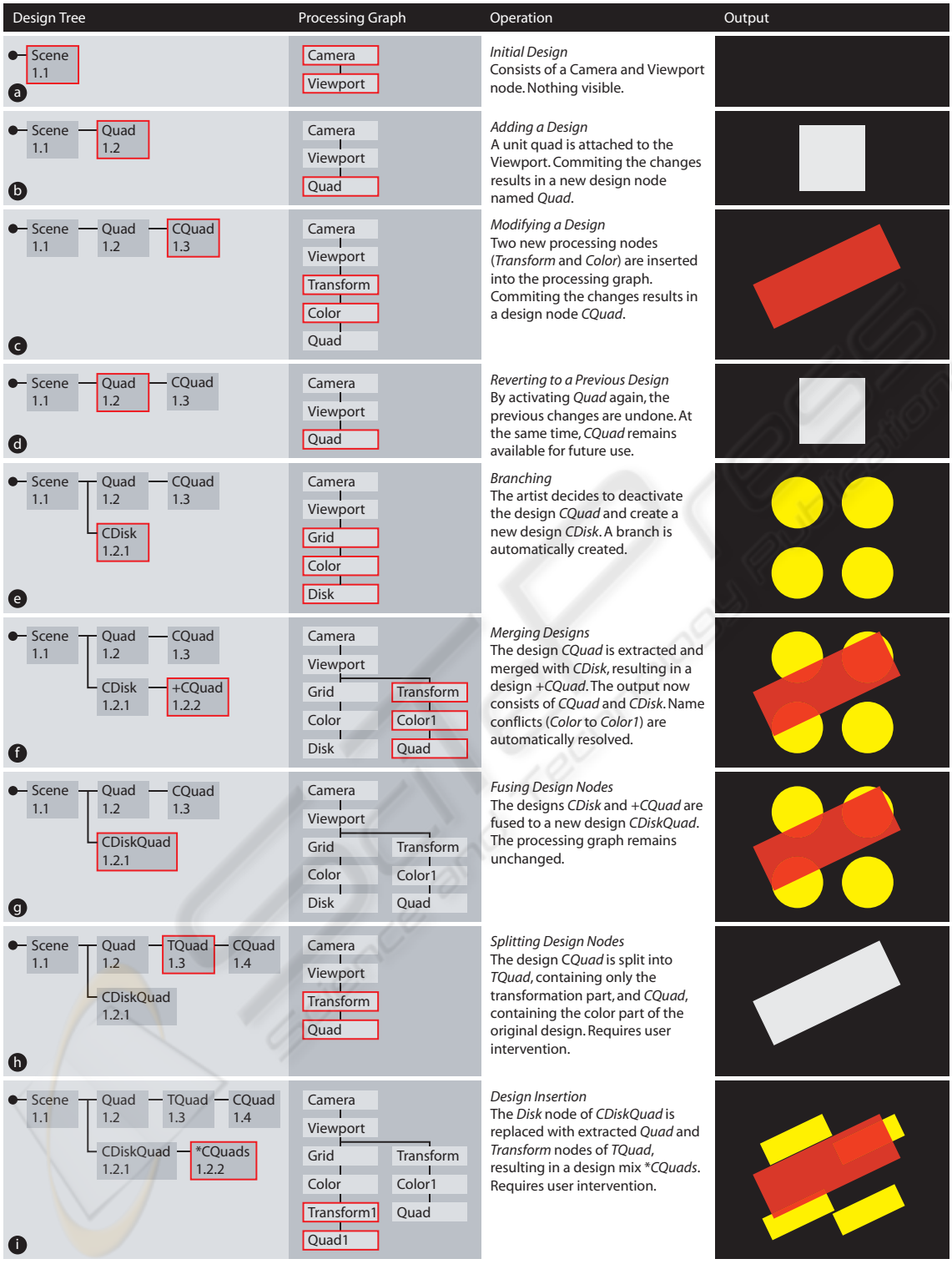

Figure 6: Design tree operations (rows) and their effects on the processing graph. The framed boxes in the design tree column refer to the currently active design node. The framed boxes in the processing graph column indicate changes evoked by activating the corresponding design node.

order has usually little importance for the final set of designs the artist wants to use during a performance. Furthermore, not every revision is necessarily a new design (Ramakrishnan and Ram, 1996). Hence, a number of high-level design operations acting on multiple design nodes complete design tree navigation and branching:

Node merging allows the user to combine arbitrary existing designs while automatically resolving identifier conflicts when nodes are merged (Figure 6-f).

Node fusion allows the user to unify two subsequent design nodes (Figure 6-g).

Node splitting allows the user to subdivide an existing design node (Figure 6-h). User intervention is required to inform the system how the design split has to be performed.

Module extraction encapsulates a subgraph of the processing graph in a module. Extracted modules are available for design insertion. User intervention is required for subgraph selection.

Design insertion allows the user to replace a processing node with a extracted module (Figure 6-i). Whereas node merging *unifies* two designs, design insertion *changes* a design by replacing a processing node by one or more processing nodes and connections. User intervention is required for module and processing node selection. The system proposes a default reconnecting scheme (e.g., maintaining parent-child relationships for scene graph connections) that can be interactively adapted.

SL2 identifier renaming is an interactive method to change names of design nodes and processing nodes. Explicit renaming is typically required after automatic name resolution of merge operations.

These high-level operations are implemented by *refactoring methods*, as known from software engineering (Fowler, 1999) applied to the SL2 code representing a design. Unlike file versioning systems, where every revision is considered immutable, the refactoring methods may change arbitrary design nodes in the tree. The operations are invoked by user interface actions.

5.2 System State Parameterization

The system does not distinguish between preparation and performance mode. However, *structural changes* to the processing graph, such as adding and removing processing nodes and connections are typically more frequent during preparation. In contrast, *parametrical changes*, i.e., assignment of values to input ports, are more common during performance. Changing multiple parameter of a design imposes a problem when switching designs during performance: Strictly following the undo/redo philosophy, all parameter changes will be reverted when deactivating a design node and visually important aspects of an artwork may abruptly change (e.g. motion or color).

In order to maintain visual consistency, the parameter spaces of SOUNDIUM may be treated orthogonal to the structural space consisting of processing nodes and connections. This is achieved by weighted parameter vectors called *valuesets*, whose scope can span multiple design nodes. Valuesets can be defined at every design node and comprise all or a subset of a design's parameters. Since valuesets only contain parameter changes, they can be applied weighted with respect to the current system state, and smooth transitions between the current parametrization and the valueset can be achieved. By changing a single weight, the user can interactively modify a potentially large number of design parameters at once.

Typical applications of valuesets are global and high-level properties of an artwork, such as "softness", "speed", or "entropy" that have general purpose application independent of the actual processing graph.

5.3 The Interactive Design Process

The interactive design process using the design tree can be split into two major stages: In a preparation phase, designs are created in terms of an upcoming performance. During performance, the predefined designs are selected and further modified during the performance. How detailed a performance is prepared is not technically constrained but rather the artist's choice, and often influenced by external factors such as the performance's duration.

Yet, a general procedure for preparation is inherent: The tree's root node generally contains a common setup, as indicated by the "Scene" node (1.1) in Figure 6. In practice, the setup includes a lot more than just a camera and a viewport. For instance, full-scene post effects (e.g., motion blur, glow, masking) and global transformations (e.g., camera movement) are typically placed in the root node, or in nodes close to the tree's root. In addition, global valuesets (e.g., overall scene color modification) are placed at this point. Consequently, every design deeper in the tree's structure will inherit the particular global configuration.

The next step is the definition of the actual designs, which may be re-used and adapted from previous performances as needed. The design tree supports the artist in structuring the performance: Designs can be arranged as needed by creating design variations (e.g., *CQuad* is a variation of *Quad* in Figure 6-c) or branches containing a different design (*CDisk* in Figure 6-e). The arrangement may serve as a temporal

plot or even as a script during performance, or assist multiple artists using the same common setup along with their individual designs.

Because several parameters (e.g., those resulting from real-time music analysis) are unknown during the preparation phase, these designs are typically anticipatory – their final visual rendering is partially unknown and only emerges in function of the environment during the live performance. This property is particularly effective for VJ performances that usually last several hours with musical content that is not predictable in advance. In addition, anticipatory designs provide enough space for improvisation during the performance.

Throughout the performance, the artist selects, fine-tunes, parameterizes, and applies designs, thereby countervailing the unknown and adding his perception of the moment. All changes to the system state (i.e., the processing graph) are tracked and can be committed to build new design nodes in order to be reused. Using the refactoring methods presented earlier an interpolation from the current design to a target design is easily achieved. This powerful mechanisms implement the idea of a *design path* (Simon, 1996) and allow for a carefully adjusted and uninterrupted performance.

6 RESULTS

SOUNDIUM has been used at numerous live performances ranging from large-scale VJ events to video art performances to interactive multimedia installations. The system has been evaluated by both novice visual artists as well as expert computer artists. For both groups we observed an immediate capability of using the media manager. On the other hand, getting accustomed to the design tree required a tutoring time of a day or two, but then resulting in a steep learning curve. While the novice artists mainly used the tree for recalling designs and as a undo/redo facility, the experts quickly explored the refactoring methods and used them to create their own designs. This section gives concrete examples that were realized interactively using SOUNDIUM.

The first example comprises a typical live performance scenario. It illustrates how the design tree and processing graph operations are applied to quickly evolve a sphere into an audio-driven "digital dance floor". The artist starts by modeling a simple 3D scene and gradually connecting it to extracted audio features: A sphere is scaled by the low frequency levels of the incoming audio signal and moves on an elliptical orbit, changing its direction on every bass drum onset (similar to Figure 3). Additionally, the bass drum onsets also change the faces' filled/wireframe state. The resulting output is shown in Figure 7 (top row). Now, the performer's design goal is to move the camera into the sphere and add a dance floor including dancers. Since this occurs during a live performance, the following steps must be executed reasonably fast and with the help of a preview output (i.e., intermediate steps are not visible on the main outputs), which is illustrated in Figure 7 (center row). The dancing girls (Figure 1) are extracted from the artist's design tree and merged with the current design. A "multiple copy" processing node is inserted to generate a crowd of dancers. The dance floor consisting of a grid of quads is then modeled from scratch through direct manipulation of the processing graph. The quads' colors are defined by both the audio level and the grid position. Then, the dance floor and the dancers are transformed to fit into the sphere. Now, all components are in place and ready to be successively shown on the main output. Ultimately, the camera is moved into the sphere and a set of audio-driven different camera viewing-angles and transformation speeds is added (Figure 7, bottom row). Altogether, above steps take the (experienced) artist about 10 minutes.

The second example illustrates how design operations (Section 5) are applied for efficient live design editing. The source design (Figure 8, left) is transformed into a target design, i.e. five cubes representing the audio spectrum (right). Two possible design transitions are shown. In both transitions, design insertion has been applied to transfer the target's background color to the source design while the source design's structural dominance has been gradually reduced. Application of the design operations including user intervention took about 5 minutes for each transition, provided that source and target design are given in advance.

In the last example, analyzed video footage has been rearranged in real-time by interactive NLE: Using the media manager, the video was automatically segmented into three scenes consisting of several shots. Scene 1 consists of five groups of shots showing a woman in blue color tones, scene 2 consists of two groups showing a close up of a man's face in dark color tones, and scene 3 consists of four groups of the woman and the man waiting for a train. During performance, the artist matches these three scenes to three different patterns detected by a dedicated "novelty measure" audio processing node (Foote et al., 2002). In addition to the computer-assisted alignment of scenes to music parts, individual shots are nonlinearly stretched in order to fit into beat intervals. Figure 9 illustrates two bars of music together with the representative keyframe of the emerging shots (the difference between musical pattern 2 and 3 is not visible in the waveform).

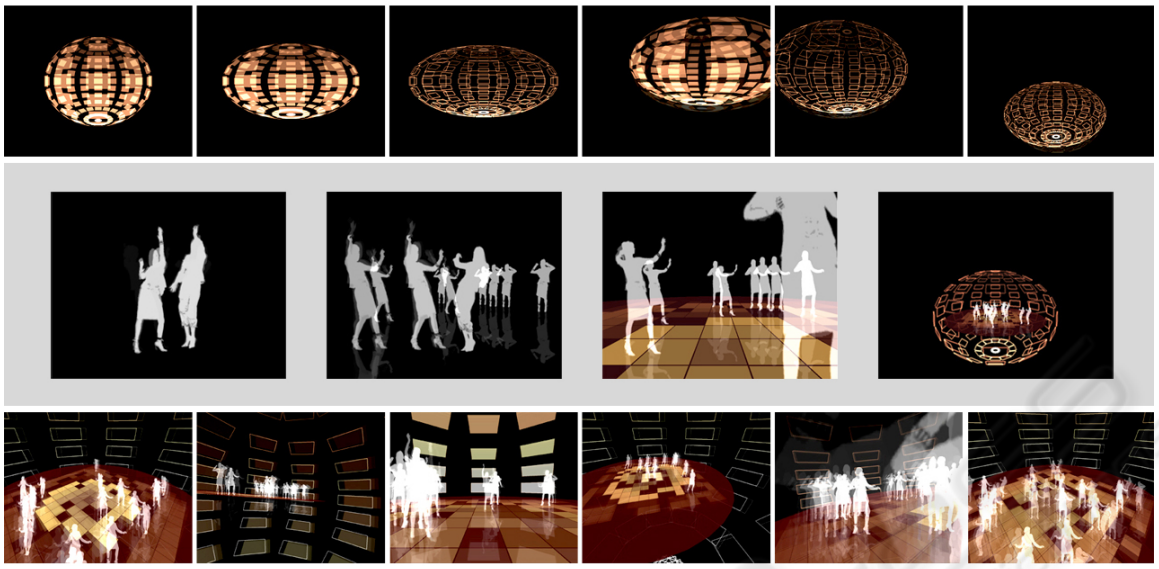

Figure 7: Designing a digital dance floor. The top row shows the current design, visible to the audience on the main output while the artist prepares the transition into the sphere on the preview output (center row). Eventually the resulting scene is shown from different camera angles that are switched in context to the music (bottom row).

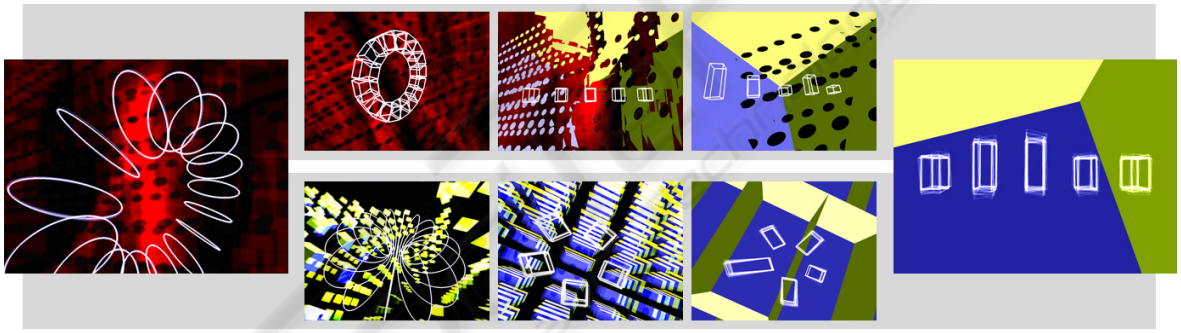

Figure 8: Design transformation example. The design on the left is transformed into the one on the right by applying design operations. Two possible alternatives are illustrated.

Figure 9: Interactive real-time video editing example. Different shots of three scenes are aligned to different music parts using similarity analysis. The individual shots are non-linearly stretched to beat intervals. The beat numbers (4/4 measure) are denoted on the horizontal axis.

7 CONCLUSIONS

We presented two novel extensions for live visual performance systems. Together, the contributions raise live performance to a higher level, allowing the artist to focus on design and content instead of low level processing details.

First, the media manager substantially enhances the way the artist interacts with large media libraries. Most important, annotated and segmented video clips together with real-time audio analysis methods can be used for real-time non-linear video editing during performance. Although the computer vision methods applied for video segmentation impose certain limitations in terms of correctness, they have proven enormously valuable, and the method of automatic analysis and manual adjustment is much faster than manual segmentation. In addition, since we adhere to the MPEG-7 standard, annotated "ready-for-use" video footage databases will eventually become available. Nonetheless, a future work direction is the implementation of additional vision methods for visual semantic-based mining and retrieval (e.g., object recognition using local features). Interactive NLE could be facilated by introducing a control grammar (Wonka et al., 2003) to encode editing techniques. Furthermore, interactive video compositing tools could be integrated into SOUNDIUM (Wang et al., 2005).

Second, the novel concept of the design tree stores and organizes the visual artist's designs and acts as a link between preparation and performance. High level design operations and refactoring methods significantly reduce the time required to create and manipulate individual designs. They further provide means of design interpolation and allow for a seamless performance. For more intuitive usage and navigation of large design repositories (e.g., thousands of designs), future work may introduce a semantic-based annotation model for the design tree.

Both extensions have been implemented as part of the SOUNDIUM live visuals system. We however believe that both of them can be applied to other existing live visuals systems. In view of ever growing media libraries it is only a matter of time until other systems *must* provide a media manager. We further believe that the design tree, which is a unique and very general concept *per se*, is not only suited for live visuals systems but could be adapted for virtually any interactive software system, in the simplest case by just replacing the common one-dimensional undo/redo mechanism.

Finally, finding an optimal user interaction model for computer-based live performance certainly remains an attractive direction for future work. In particular, we focus on live composition and performance of music *and* visuals using the same software system.

ACKNOWLEDGMENTS

We thank Jürg Gutknecht and Luc Van Gool of ETH Zürich, and Christoph P. E. Zollikofer of University of Zürich for supporting our work. Thanks also to Philippe Wüger, Mortiz Oetiker and David Stadelmann for implementing the vision algorithms. This research was supported in part by the NCCR IM2, by SNF grant 205321-102024/1, by Swisscom, and by the mighty Corebounce association.

REFERENCES

Brossier, P., Bello, J. P., and Plumbley, M. D. (2004). Real-time temporal segmentation of note objects in music signals. In *Proceedings of the 2004 International Computer Music Conference*. International Computer Music Association.

Cederqvist, P. (1993). *Version Management with CVS*. Signum Support AB.

Eisenstein, S. (1994). *Selected Works 2: Towards a Theory of Montage*. British Film Institute. Ed. by Nichael Glenny and Richard Taylor.

Foote, J., Cooper, M., and Girgensohn, A. (2002). Creating music videos using automatic media analysis. *Proc. ACM Intl. Conf. on Multimedia*, pages 553–560.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Object Technology Series. Addison-Wesley.

GarageCube (2005). Modul8 real-time video mixing and compositing software. http://www.garagecube.com.

Gorbmann, C. (1987). *Unheard Melodies: Narrative Film Music*. Indiana University Press.

Goto, M. (2001). An audio-based real-time beat tracking system for music with or without drum sound. *Journal of New Music Research*, 30(2):158–171.

Jehan, T., Lew, M., and Vaucelle, C. (2003). Cati dance: Self-edited, self-synchronized music video. In *GRAPH '03: Proceedings of the SIGGRAPH 2003 Conference on Sketches & Applications*. ACM.

Lienhard, R. (1999). Comparison of automatic shot boundary detection algorithms. In *Image and Video Processing VII*. Proc. SPIE 3656-29.

Manjunath, B. S., Salembier, P., and Sikora, T. (2002). *Introduction to MPEG-7: Multimedia Content Description Interface*. John Wiley and Sons.

Pukette, M. (2002). Max at seventeen. *Computer Music Journal*, 26(4):31–43.

Ramakrishnan, R. and Ram, D. J. (1996). Modeling design versions. *The VLDB Journal*, pages 556–566.

Rui, Y., Huang, T. S., and Mehrotra, S. (1999). Constructing table-of-content for videos. *ACM Multimedia Systems*, 7(5):359 – 368.

Scheirer, E. (1998). Tempo and beat analysis of acoustic musical signals. *J. Acoust. Soc. Am.*, 103(1):588–601.

Schubiger, S. and Müller, S. (2003). Soundium2: An interactive multimedia playground. In *Proceedings of the 2003 International Computer Music Conference*. International Computer Music Association.

Simon, H. A. (1996). *The Sciences of the Artificial*. MIT Press.

Strauss, P. S. and Carey, R. (1992). An object-oriented 3D graphics toolkit. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, volume 26, 2, pages 341–349. ACM.

Sutherland, I. E. (1963). *Sketchpad, a Man-Machine Graphical Communication System*. PhD thesis, Massachusetts Institute of Technology.

Tseng, B. L., Lin, C.-Y., and Smith, J. R. (2002). Video personalization and summarization system. In *Internet Multimedia Management Systems*. SPIE Photonics East.

Ulyate, R. and Bianciardi, D. (2002). The interactive dance club: Avoiding chaos in a multi-participant environment. *Computer Music Journal*, 26(3):40–49.

VJCentral (2005). http://www.vjcentral.com.

Wagner, M. G. and Carroll, S. (2001). Deepwave: Visualizing music with VRML. In *VSMM '01: Proceedings of the Seventh International Conference on Virtual Systems and Multimedia (VSMM'01)*, page 590. IEEE Computer Society.

Wang, J., Bhat, P., Colburn, A., Agrawala, M., and Cohen, M. (2005). Interactive video cutout. In *Proceedings of ACM SIGGRAPH 2005 / ACM Transactions on Graphics*, volume 24, 3, pages 585–594. ACM.

Wonka, P., Wimmer, M., Sillion, F., and Ribarsky, W. (2003). Instant architecture. In *Proceedings of ACM SIGGRAPH 2003 / ACM Transactions on Graphics*, volume 22, 3, pages 669–677. ACM.