Future Visuals: a Real-Time 3D Graphics Synthesizer for VJ Performances

Lieven van Velthoven, *Media Technology, Leiden University*

***Abstract*—Based on the observation that VJ performances still mainly exploit video loops, while gaming and *Demoscene* technologies are starting to provide near-photorealism in real-time, we formed the idea that real-time 3D graphics techniques are *at least as* suited for VJ work as traditional loop-based ones. To assess this idea, we developed and evaluated a novel software tool: a visuals synthesizer for live performances which can generate dynamic, highly customizable 3D graphics based on point cloud data.**

We describe the design of our software and the rationale behind it, and provide an assessment of the system's strengths and weaknesses. A short summary of reactions from a series of live performances is also presented.

***Index Terms*—human computer interaction, VJ performance, visual music, real-time computer graphics**

I. Introduction

THIS document describes the development of a software project that was motivated by *Demoscene* works on the one hand, and VJ performances on the other. The project was based on the following observations:

[1] VJ culture is still (mostly) based on 2D video loops.
[2] Gaming technology is starting to provide near-photorealism in real-time.
[3] 'The Demoscene' has been synchronizing real-time visuals to music for decades.

Combining these three observations, the concept of the project is that real-time 3D graphics techniques from the game and demo scenes should be *at least* as suited for VJ work as popular 2D loop-based ones. The reason for this is multifold. Firstly, since all graphics are being generated live, *everything* about the visuals can be adjusted to correspond with live music. This is not the case with pre-recorded or pre-rendered content, where much of what you see is more or less fixed (e.g. camera framing, shapes, movement, etc.).

Also, a fully software-based system can be automated to any possible extent. Besides relieving the VJ of a lot of work, it could eventually even replace him or her. In theory, it could outperform any human VJ in terms of responsiveness; i.e. complexity and latency of the visuals' reactions to live music.

Lastly, a 3D graphics synthesizer could still fully incorporate 2D loop-based visuals as well. In fact, gaming and demo technology is perfectly suited for rendering complex, composite video effects even *while* rendering 3D content, as many games and demos show.

For the reasons just described, we decided to design novel VJ software in the form of a real-time 3D graphics synthesizer for gaming notebooks: an instrument to generate aesthetically interesting visuals, operated by a performer and/or reacting to live music. The goal was to have a tool that is good enough to be 'tested' by the authors in real-life night club scenarios.

We will explain the main design and features of our software in this paper, and provide an evaluation of the system's strengths, limitations and reception.

II. Related Work

VJ tools

As mentioned, most VJ performances today rely on the editing and mixing of video footage and the use of visual effects, filters and feedback. This can be seen by inspecting the main features of popular VJ software tools such as Resolume [10], ArKaos [11], Modul8 [12], etc., which are often used in tandem with Adobe After Effects [13] or Quartz Composer [14] as a source for original source footage. Real-time (3D) graphics creation and control functionality is often quite limited in VJ tools (i.e. basic geometric solids with (video) textures and/or effects). This is surprising if you consider the great results of real-time graphics in videogames and other applications, but it might be explained by two personal observations from VJ-ing experience:

1. VJs are more often from art, graphic design or animation backgrounds than from computer science or computer graphics (CG) backgrounds.
2. Among VJs, Apple's MacBooks are by far the most commonly used computers. However, their 3D graphics capability is limited compared to same or lower-priced gaming PC's, which tend to compete strongly this field.

Manuscript draft: August 30, 2012. This document was formatted in compliance with IEEE standards as described in "Preparation of Papers for IEEE TRANSACTIONS and JOURNALS (May 2007)". The author does not permit publication of this document in any way or form.

Lieven van Velthoven was an MSc. student at the Leiden Institute For Advanced Computer Science, Leiden University, the Netherlands at the time of writing (e-mail: lievenvv@gmail.com).

The Demoscene

'The Demoscene' does not enjoy much attention from the general public although it has gathered a dedicated following over past decades. The Demoscene, or just 'scene' for insiders, is a community of people that create and share 'demos': real-time, non-interactive audiovisual presentations in the form of executable software [17].

Many 'demo parties' are organized around the world every year, where demo groups come to show their creations and compete on style and skill in various categories on several computing platforms. (refs parties). Often, these demos show impressive graphics in tight synchrony to music: all running in real-time on affordable hardware.

A bibliography of publications about the Demoscene can be found here: http://www.kameli.net/demoresearch2/?page_id=4

Other attempts to join Demoscene and VJ cultures have been made in the (recent) past, often with good results. Collectives like Pixelsync, Anti-VJ and [Aphex Twin's VJ][refs] have successfully exploited live 3D graphics in their shows. However, not all of this work has been documented, and they each have different approaches than our own.

During our research we identified some current trends regarding innovation in the field of VJ culture. One is to shift focus towards the VJ him/herself as a performing entertainer, e.g. through the use of gestural interfaces. Another is to try and increase audience participation and feedback, by means of (wearable) sensors, cameras, mobile devices and/or other means. A last trend we noticed concerns creativity in stage design and/or display screens, e.g. multiple, or custom shaped projection screens and projection mapped sculptures. See e.g. [2],[6],[8].

The above approaches do not directly enhance the aesthetic and synergetic quality of the actual visual output. Although they are interesting directions in their own right, this project focuses instead on the main modality of VJ work: the quality and synchrony of visuals.

III. Project Description

A. *Point cloud graphics*

In order to limit the scope of the project, we focused on exploring particle-based graphics while developing our prototype VJ-software. Although we plan to add shaded polygon graphics functionality in the future, it is the current standard in computer graphics and the results should therefore be predictable. In addition, points (and line) graphics present a unique abstract style that still allows for extremely highly detailed scenes and graphics. Also, particles can be animated quite easily by means of common physics simulation techniques. This allows for the generation of parameterized motion with inherent aesthetic qualities, without accessing the skills of any animators. An illustration of the aesthetic qualities of particle animations can be seen in the current popularity of Krakatoa [ref] and FumeFX [ref] in commercials and 'leader clips' on TV and the web.

We used point clouds from various online resources, ranging in size from several hundred thousand points to over fifty million. We also added functions to generate point cloud data from 2D images and (live) camera footage, through simple thickening and intensity-based false depth, respectively. Real-time 3D scanning of the DJ or crowd was made possible through thankful use of Microsoft's Kinect Sensor.

B. *Color maps*

The primary source of color in our system is formed by a set of user-created, one-dimensional '*color maps*'. When a point cloud is imported, each particle gets a color index based on image/laser scan intensities or Kinect depth data. During rendering, this index is looked up in a user-selected color map, thereby always ensuring that only 'pretty' and well-matching colors get used as a basis for the visuals.

C. *Parameters*

In order to create customizable graphics from point cloud datasets, we needed ways to interactively control the rendering. We started by defining the main elements of our visuals synthesizer, simply called *parameters*, as follows:

Parameter:

Fixed properties (pre-defined for each parameter):

- *name:* a user-friendly name for the parameter.
- *action:* the function that this parameter controls.
- *range:* the absolute minimum and maximum values that the parameter can take on.
- *default value:* a default (starting) value for the parameter.
- *ramp type:* defines how the parameter's value changes with respect to the normalized control value: i.e. linearly, quadratically, sub-linearly or according to a custom user-defined ramp.

Adjustable properties (adjustable by user or automation):

- *value:* the normalized, current control value for this parameter.
- *control type:* determines how the parameter is currently being controlled:
  - manually: by the user.
  - automatically: by a '*signal generator*'.
  - automatically: by recorded user input.
  - automatically: by features of live audio input.
- *animation range:* the currently desired value range for automatic parameter control.
- *paused:* (un)pauses any automatic control of the parameter.

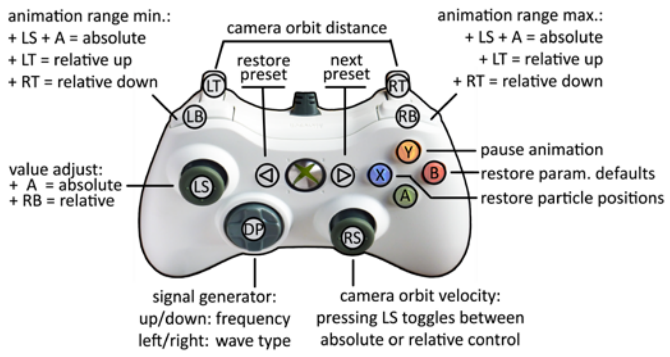


Fig. 1: button mapping for Xbox or PlayStation game controllers.

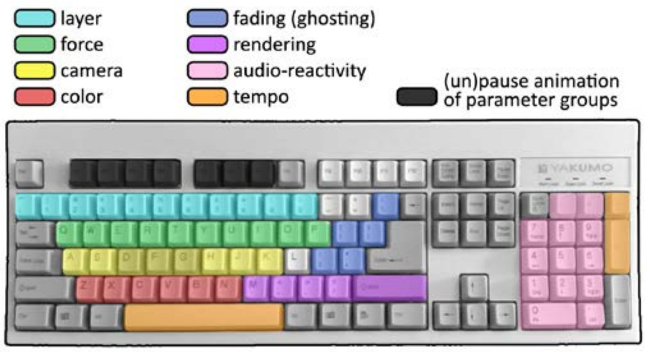


Fig. 2: button layout showing parameter groups on a standard PC keyboard.

D. *Actions*

Each of these parameters governs its own '*action*'. Many basic options and parameters offered by DirectX were mapped to these actions: e.g. primitive drawing type, alpha transparency, blending mode, camera position, texture maps, et cetera. Some more advanced functions were created especially for the system, such as post-processing effects and animation.

To allow quick, direct selection of parameters for user adjustment, all parameters were mapped to single keys on the computer's keyboard, and related parameters were grouped onto neighboring keys.

E. *Parameter adjust*

After selecting a parameter on the keyboard any of its non-fixed properties can be modified on the-fly. The parameters' values and ranges can be controlled relatively and manually, both manually and automatically.

We looked at a number of input devices to determine which would be appropriate for our software during live performances. The computer mouse provides only 2 independent axes and a small number of buttons, while being somewhat impractical and unreliable on cluttered night club stages. Game controllers (e.g. the wireless ones of the Xbox and PlayStation consoles) proved much better because of their larger number of buttons and axes, their great portability and low space requirements. Midi controllers, in various forms and shapes, can also form ideal input devices for parameter selection and adjustment. Depending on the device, they can require some table space though: a luxury that is not always available to VJs.

Our software's current button mapping for keyboard and game controller is illustrated in figures 1 and 2.

F. *Signal generators*

The first type of automatic parameter control we implemented was a class of simple '*signal generators*', providing signals for the parameters' control values. The user can choose between several '*wave forms*', or '*record*' one, and set the '*phase*' and '*frequency*' of the wave's repetition. The frequencies are all powers of two with respect to the music's '*tempo*'. As a provisionary, yet robust way of defining the tempo, it is (currently) set by the user through tapping along on the spacebar.

G. *Physics simulation*

For this experiment we implemented a provisional, basic system of springs, drag, and spherical forces, which runs on the graphical processing unit. The currently offered parameters include *simulation speed* (the time step of the simulation), *spring strength (*makes particles return to their original position in the point cloud), and *drag (*the rate at which particles slow down over time).

Simulated forces cause the particles to move. In our prototype system, they are limited to spherical force fields that randomly change their position, within adjustable *ranges* and at user-defined *intervals* (e.g. once every musical 'beat' or 'bar'). Their *attractive/repulsive strength*, *number*, and *radius* can also be adjusted: i.e. are also linked to parameters.

H. *Layers*

After debut performances in home and night club settings, another main feature to be added was an implementation of multiple independent '*layers*'. This greatly increased the complexity and flexibility offered by our visuals synthesizer. Experiments showed that, running on a mid-range gaming laptop, there was enough computational power to draw several multi-million point clouds, albeit with physics simulation limited to just one. We added multiple independent, but identical sets of parameters that each control superimposed layers of graphics. A total of ten independent layer slots were chosen to be available, limited by practical hardware capabilities and keyboard space.

I. *Post-processing*

Several image post-processing and backbuffer feedback effects were implemented, such as image blur, Sobel edge gradient, posterization, lens effects, pixelation, ghosting, et cetera. Most of these effects are independent between layers: e.g., the edges of a logo can appear before a pixelated 3D scene with blurred graphics in the background.

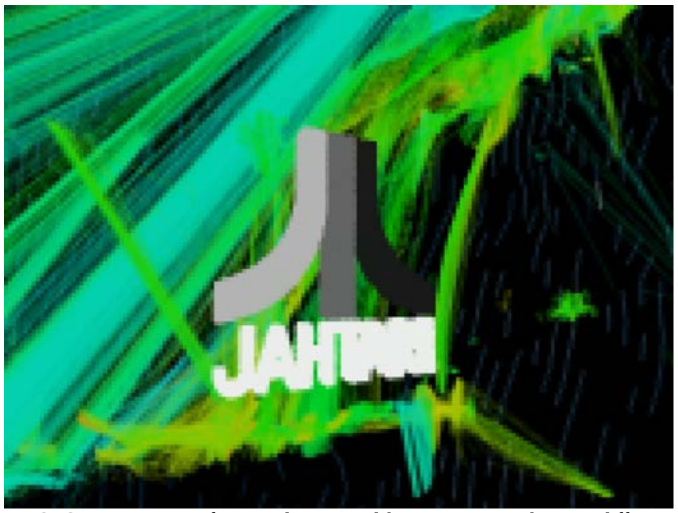


Fig. 3: Screen capture of a visual generated by our system, showing different layers and an (audio-reactive) pixelation effect.

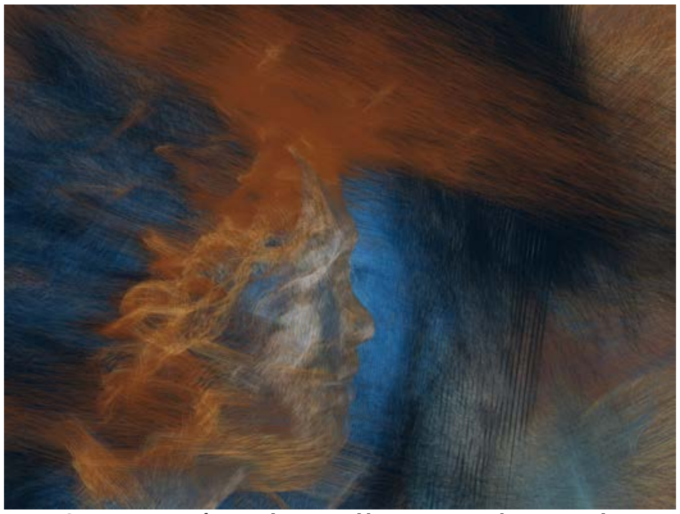


Fig. 4: Screen capture of a visual generated by our system, showing an almost painting-like 'look'.

J. (Sub-)presets

The features described above provide full functionality for WYSIWYG, live creation of visual music. Since, obviously, some combinations of parameters will look 'better' than others, we added the possibility to store all the synthesizer's current settings in so-called 'presets'. This allows the VJ to prepare for a set by building a collection of good-looking presets in advance. Later, we also implemented what we call 'sub-presets', which can store any combination of parameters' settings. They can be applied to one or more layers, instantly adjusting a certain set of parameters in some desired way. For example, you could store a complex camera movement pattern, or a specific combination of audio-reactive color and post-processing effects. This allows artists to design and re-use 'visual motifs'.

K. Graphical User Interface

A Graphical User Interface (GUI) was designed for display on the main screen of the VJ's laptop. Initially, it offered graphical feedback and interaction for each of the parameters. This was later sacrificed to make space for other functions, partly because the GUI proved not as fast and accurate for input as the keyboard and controller. The visuals in our software are mainly created using its "what you see is what you get" (WYSIWYG) interface; the GUI offers 'extra' functions, i.e. to load point clouds from disk and assign them to layers, and to facilitate the use of presets and sub-presets. It also shows the color maps and offers various other functions, such as recording and playback of Kinect data. While providing useful functions, a '*freestyle*' performance could still be given without it.

IV. Evaluation

We will describe the strengths and limitations of the system, and our personal reflections on using it. We will summarize the feedback received from crowds, party organizers and fellow VJs, having performed with the system at various well-known clubs in the Netherlands. Due to the early development phase of the project, no other artists have yet performed with our software.

A. Strengths of the system

Aesthetically, the software can produce dynamic, 'high-tech' looking visuals (please see figures 3-9 and the footage in [15],[16]). Diverse effects can be achieved by combining different parameters, e.g. slow or fast, dreamy or rough, abstract or figurative, colorful or black and white, et cetera. Due to the large number of parameters, the possibilities to animate the visuals and synchronize them to music are vast. The audio-reactivity can produce intricate visual music when distinct 'visual motifs' are assigned to 'beats' of different frequencies: e.g. representing the bass drum, snare drum and hi-hat.

The physics simulation delivers organic, mesmerizing motion, which is easily adjusted through its parameters to create different looks: tight or smooth, rhythmic or continuous, subtle or intense, noisy or crisp. Also, scenes can be (rhythmically) morphed from one into another. For example: live, 3D Kinect 'footage' can change into a logo and back once every eight beats, while forces distort it on every other beat.

On the technical side, our system also has some benefits over video-based VJ software. First of all, the renderings it produces are always pixel-perfect and without any video compression artifacts. Moreover, it can do so with complete flexibility towards display resolution and aspect ratio. This is not possible with video-based approaches in general.
Besides the benefits of real-time rendering, the software also has considerably lower hard disk space requirements than loop-based VJ software; the memory needed to store 1 million uncompressed points can only hold 2 or 3 frames of uncompressed HD video. Also, our system relies almost

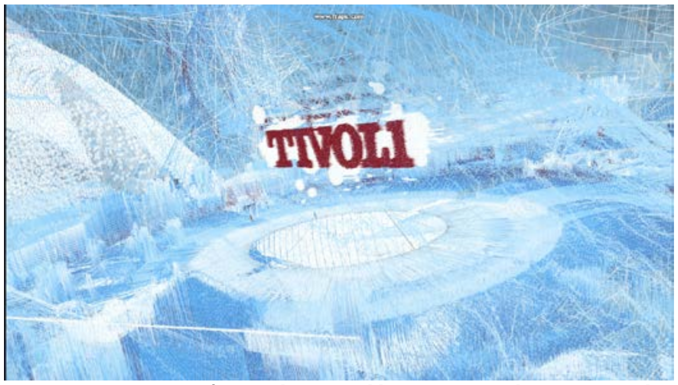


Fig. 5: Screen capture of a visual generated by our system, showing a complex, 'icy' look. This rendering comprises 6 layers of each >1M points; and yielded a very strong sense of depth when seen in motion.

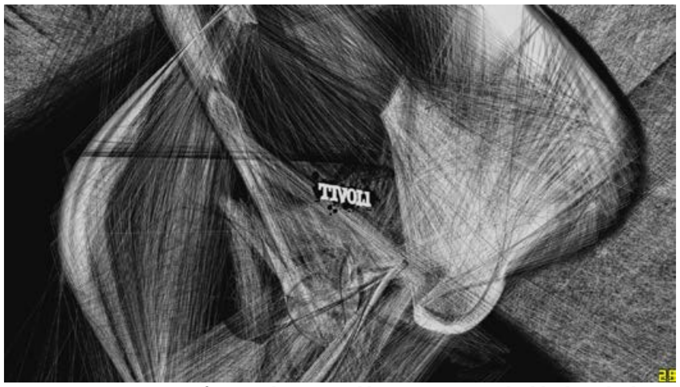


Fig. 6: Screen capture of a visual generated by our system. The black and white lines where moving in reaction to simulated physics forces, and were continuously rendered over the last frame's contents.

completely on the graphics processor (GPU), and leaves the computer's central processor (CPU) almost idle. This means that a GPU upgrade will directly raise the limits of graphics detail and complexity, but also that plenty of CPU time is left for additional, future functionality of the software.

On the user control side, a keyboard and game controller combined with audio-input proved minimally sufficient for complete live performances. While being extremely portable, this combination allows for fast and direct access to all parameters. Audio-input affords strong audiovisual synergy, linking graphics and sound much more directly than would be possible without it.

B. Limitations of the system

Quite diverse graphics can be produced with our system, but due to the particle / line rendering the style is still relatively distinct and recognizable. Video-based and 3D non-point cloud graphics features are still desired for enhanced artistic freedom, but were outside the scope of this project. Nonetheless, they should be trivial to implement using commonly available software libraries: e.g. to import animated 3D models and 2D videos for live synchronization. Also, higher level or combined parameters are still welcome to further enhance controllability and complexity, as are advanced triggering and sequencing features.

The system has a relatively steep learning curve; users need to get very familiar with the system to get the best results. Nevertheless, this is the case with any (complex) synthesizer, and free experimentation can quickly lead to pretty pictures. These can then be saved for future performances, allowing even 'beginning' visual artists to prepare interesting VJ sets.

Since the graphics are generated in real-time, they are limited by the hardware's capabilities. Many things that can be done with pre-rendered graphics are simply impossible to do in real-time at the moment. In our software, some (very) specific combinations of parameters can lead to slow-down as well, although this is easy to prevent or intercept by the system. Still, gaming grade hardware is (more than) capable of producing interesting visuals, and has been developing at a fantastic rate. Also, the benefits of real-time rendering for live performances may well outweigh the benefits of pre-rendering: especially considering the fact that 'cinema quality' animation is outside the scope of most VJ work anyway.

Another 'limitation' of our system is actually due to an unsolvable problem: not *every* perceptive property of sound can be detected without noticeable delays. This is a general problem in signal processing and VJing. Detecting 'beats' is no problem, but some other or higher level properties like 'bridges' or 'drops' in music, are. On the other hand, human brains face these same problems: i.e. the visuals would react to a 'drop' at roughly the same time as the crowd.

A final weakness of our software is a more practical one: the available keyboard space for parameter selection is getting very limited. If many more parameters are implemented, direct parameter selection will become difficult.

C. Feedback from real-life performances

We performed with our software at various real-life club settings in the Netherlands, throughout the development process of our software. After showing them early footage of our visuals, *Vol Producties* invited us to perform at a sold-out party in LVC, Leiden. The response from both the crowd and organizers was very positive; we got invited again several times. Another series of performances was done for *Bassculture*: performing in well-known venues like *OT301* and *het Concertgebouw* in Amsterdam. The *DEFRAME Collective*, a successful group of VJs, were so impressed that they let us perform for the headliner DJs at festivals in *TrouwAmsterdam* and *Nemo Science Center*, and invited us to do the same at an upcoming festival in Blackpool, UK. Another successful Dutch VJ collective, *WERC*, also expressed their appreciation and invited us to join them for a 'back-to-back' VJ session during one of their performances. In practically all cases, we received many spontaneous compliments from the crowds as well.

V. Conclusion

We described the design and rationale behind our VJ visuals synthesizer software, and the main strengths and limitations of our approach. The real-time, three-dimensional, point cloud based graphics can yield a wide range of dynamic, highly detailed visuals and seem to fit the abstract nature of electronic dance music quite well. Although the visual style is somewhat recognizable due to the current restrictions on our graphics, many different visual ‘moods’ can still be created with the system and without ever ‘looping’ the graphics (unless you want them to).

The software was tested in various, real-life club settings: to guide the development process and provide feedback on the general concept of the project. These tests, or rather performances, suggested a positive reception of the software among crowds and party organizers, but also among fellow VJs. The reader is invited to form his or her own opinion by watching videos of the project on YouTube, e.g. [15],[16].

The results booked during this exploratory project encourage us to continue our work. Some aspects of the system could be further improved upon, i.e. the graphics, audio-analysis and automation, and many new features could be implemented. These may include 2D video-based graphics, 3D shaded polygon graphics, integrated 3D scanning, and video mapping functionality. Although we plan to include ‘standard’ rendering methods, indications exist that point-based rendering *might* actually become the new standard [9].

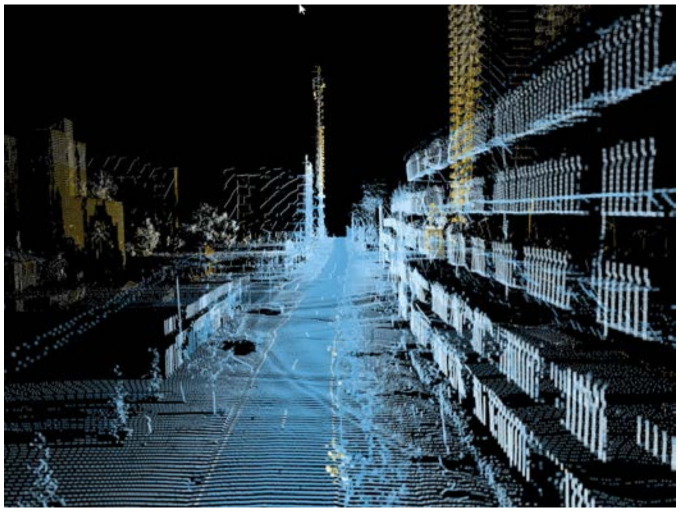
Acknowledgment

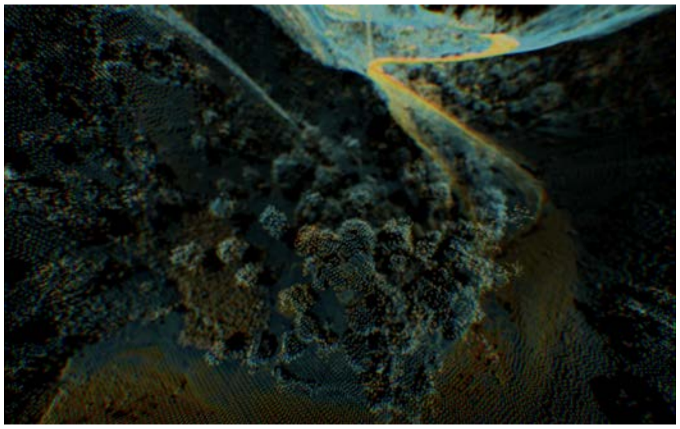
Great thanks goes out to *Radiohead* and institutions such as *Cyark.org*, *USGS*, *Opentopography.org* and many others for providing those incredible LIDAR scans.

We would like to thank *Vol Producties* for inviting us to perform at their ever-sold-out parties in LVC Leiden and the *Bassculture* crew for inviting us to some great venues in Amsterdam. We appreciate the support of the *WERC* VJ collective, and thank *DEFRAME Collective* and *FIBER* for offering us to VJ for their festivals’ headliners.

References

[1] B. Evans, "Foundations of a Visual Music", *Computer Music Journal*, Winter 2005.
[2] G. Kaiser, G. Ekblad, L. Broling, "Audience participation in a dance-club context: design of a system for collaborative creation of visuals".
[3] A. Engström, M. Esbjörnsson, O. Juhlin, “Nighttime visual media production in club environments”.
[4] A. Dekker, "Dancing in the light of an information overload", *ISEA 2004*.
[5] A. Dekker, "Synaesthetic Performance In The Club Scene", *Proceedings of the 3rd Conference on Computational Semiotics for Games and New Media*, Sep. 2003
[6] S. Taylor, S. Izadi, D. Kirk, R. Harper, A. Garcia-Mendoza, "Turning the Tables: An Interactive Surface for VJing"
[7] K. Motomura, "VJ Towards Media Art – A Possibility of Interactive Visual Expression"
[8] A. Banerjee, A. Girouard, J. Burstyn, R. Vertegaal, "WaveForm: Remote Video Blending for VJs Using In-Air Multitouch Gestures", CHI Work-in-Progress, May 2011.
[9] G. Lynch, “Euclideon Unlimited Detail vs. Atomontage: The photo-real future of gaming graphics?”, *TechDigest.tv*, Aug. 2011. Available: www.techdigest.tv/2011/08/euclideon_unlim.html
[10] Resolume VJ Software - Live Digital Motion Graphics. www.resolume.com.
[11] ArKaos - VJ Software, media server, video mixing, LED software. www.arkaos.net.
[12] Modul8 VJ software. www.modul8.ch.
[13] Adobe After Effects | Visual effects, motion graphics software. www.adobe.com/AfterEffects.
[14] Quartz Composer visual development environment. developer.apple.com/technologies/mac/graphics-and-animation.html
[15] Real-time 3D VJ visuals – Demo Reel #1. www.youtube.com/watch?v=PPO6qj3_-Wk
[16] Real-time 3D VJ visuals – FIBER Festival www.youtube.com/watch?v=lSmkmuNEt60
[17] V.M. Heikilla, “The Future of Demo Art: The Demoscene in the 2010s” (online essay), Sep. 2010.





Figures 7-9. More screenshots of our visuals, showing surreal figurative renderings of land- and cityscapes, plus a DJ at work.