The Visual in Mobile Music Performance

Patrick O'Keefe
University of Michigan
Electrical Engineering: Systems
1301 Beal Avenue
Ann Arbor, Michigan 48109-2121
pokeefe@umich.edu

Georg Essl
University of Michigan
Electrical Engineering &
Computer Science and Music
2260 Hayward Street
Ann Arbor, Michigan 48109-2121
gessl@eecs.umich.edu

ABSTRACT

Visual information integration in mobile music performance is an area that has not been thoroughly explored and current applications are often individually designed. From camera input to flexible output rendering, we discuss visual performance support in the context of urMus, a meta-environment for mobile interaction and performance development. The use of cameras, a set of image primitives, interactive visual content, projectors, and camera flashes can lead to visually intriguing performance possibilities.

Keywords

Mobile performance, visual interaction, camera phone, mobile collaboration

1. INTRODUCTION

Although mobile device interaction is tremendously visual, they inherently suffer from a limitation on screen real estate. However, this restriction is mitigated by the growing popularity of tablet devices and portable projectors; there are even some mobile phones on the market with integrated pico projectors. This indicates a general consumer interest in transcending these visual limitations and making the mobile experience more communal.

The purpose of this paper is to make the visual modality an accessible part of mobile music performance. This includes both the built-in cameras as sensor input as well as the screen and projected images as output. When incorporated into a flexible graphics and data-flow engine, it becomes possible to rapidly develop performances that seamlessly integrate computer vision, sound synthesis, and rich visual output. With the use of many mobile devices with projectors, visual display becomes more modular and a coordinated effort. The relative position of performers and the choice of projectable space expand what mobile performances are even conceivable.

For our implementation of these visual concepts, we have worked in the context of urMus, a meta-environment of mobile device programming for artistic purposes [7, 9, 8]. The goal of urMus is to make the design of all aspects of mobile phone interactions and performances easy and flexible at the same time.

2. RELATED WORK

The use of the camera for mobile phone interaction has been explored extensively, but not in a musical context [21, 20, 10, 19, 24, 18, 17]. There have also been non-mobile studies on mapping computer vision features to sound [13, 3]. In a musical context, phone cameras have been used for motion detection when mobile phones at the time did not have built-in accelerometers or other motion sensors [22, 23]. In one paper, the mobile phone was played as a wind instrument using the microphone as the wind sensor. Covering the camera (detectable by overall brightness) would act as closing a tone-hole in the wind instrument [12]. A predecessor to urMus, SpeedDial, was a Symbian mobile synthesis mapping environment which used the camera as an abstracted sensor and allowed overall camera brightness to be mapped to control a range of synthesis algorithms [6]. Mobile music making itself is an ongoing topic of interest [26, 25, 2].

There have also been several studies into the new types of interaction and experiences provided by coupling a portable projector with mobile devices [27, 15, 1]. Work by Cao has explored multi-user portable projector interaction and different types of projectable spaces [5, 4].

3. VISUAL INPUT

Digital cameras on contemporary mobile devices have high image quality and offer very fast rates of capture. One can interpret the information provided by the camera as literal – images that represent a world are to be interpreted and displayed as presented – or this information can be abstracted and used to drive performance. The goal of this work is to explore both options.

In order to enable each interpretation, we give access to camera information though two possible routes. One method is a part of a data stream pipeline where camera images are reduced to single numbers which in turn can be used to control sound synthesis. Rather than using detailed visual information, broad features of the camera image are used to provide control. The second method is access to the full camera image itself. This data is accessible via a rich OpenGLES-based rendering system that can be used to create new and diverse visual content. In this section, we will discuss how the information from the camera is received from the operating system and processed.

3.1 Access to Video Data

For iOS devices, official APIs to access video data were made available with the iOS 4.0 software update in the AV-Foundation Framework. Since our current implementation is only for these devices, this is what will be discussed here. Upon application launch, an AVCaptureSession is created for the rear-facing camera with a request to process fifteen frames per second. Most importantly, the AVCaptureSession is configured to process these frames asynchronously

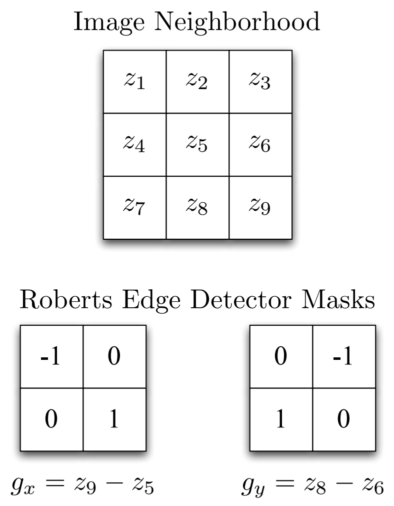

Figure 1: The Roberts Edge Detector masks and the first order derivatives they approximate.

on a secondary dispatch queue. This ensures that the user interface and other signal processing tasks (such as audio output) are not interrupted. Moreover, the secondary dispatch queue drops late video frames when the system cannot handle the requested fifteen frames per second. In practice, this happens quite often but is completely transparent to the user.

There are also three configurable aspects of the video capture process. The first is camera selection. Most iOS devices only have a rear-facing camera, but the iPhone 4 and fourth-generation iPod Touch have an option to select the front-facing camera as well. Currently, it is not possible to get data from both sources at once. The other two aspects allow the choice between an automatic or fixed setting for both the white balance and exposure. This has interesting implications for certain low-level visual features. For example, if overall brightness was being used to drive the frequency of a sine oscillator, a fixed white balance and exposure would be necessary to achieve a low frequency when the camera was covered and a high frequency when pointed at a light source. However, automatic white balance and exposure settings would result in the sonification of these processes – something that could be desirable.

3.2 Visual Features

There is no standard set of visual features that are applicable to performance situations. In the context of the urMus environment, features need to be expressed as a floating point value (or array of values) between negative one and one. To maintain generality and for computational considerations, the features developed are low-level in nature.

The first four features are overall brightness, red sum, blue sum, and green sum. Their computation is nearly self-explanatory. For a pixel buffer with n rows and m columns, the overall brightness is computed as follows.

$$\text{brightness} = \frac{1}{3nm}\sum_{i=1}^{n}\sum_{j=1}^{m}(\text{red}(i,j) + \text{green}(i,j) + \text{blue}(i,j))$$

The image processing community has developed many ways to quantify color in an image. Sometimes this involves different color spaces, color independent of brightness, and the biological processes behind color perception. Interesting situations arise: should a bright white wall have a higher "red" feature than a rose petal? For simplicity, we simply sum the components in the respective RGB channels and divide by the maximum. As mentioned above, having a locked white balance and exposure for these four features is most likely desirable.

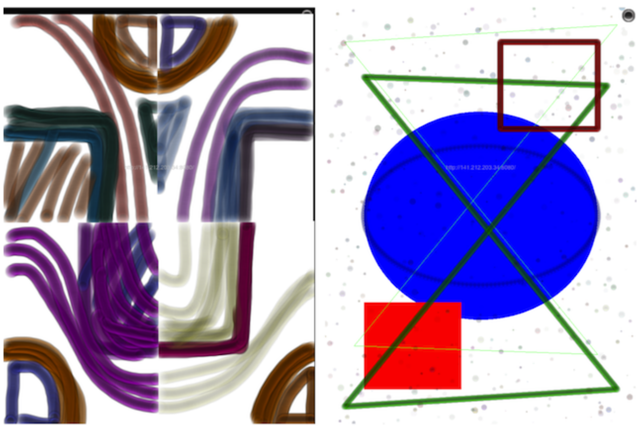

Figure 2: Examples of 2D rendering in urMus: A tiled canvas drawing program (left). The text image showing the standard drawing elements (right).

Another feature is simply named "edginess." Like the color sums, there is no standard image processing technique to determine how edgy an image is. For our implementation, the Roberts Edge Detector is used to approximate the first derivates between adjacent pixels. Figure 1 shows the masks and corresponding first order derivative approximations for this detector. The Roberts detector is one of the oldest edge detectors and is frequently used in embedded applications where simplicity and computational speed are paramount [11]. The final "edgy" feature is the normalized sum of the gradient magnitude at each pixel location where the gradient magnitude of a 2D function $f(x, y)$ is given by

$$\nabla f = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \approx \sqrt{g_x^2 + g_y^2}$$

Other low level image features can be easily incorporated into urMus at this stage. Certain features, such as optical flow, have already been investigated for mobile devices and would be natural for inclusion [22]. Higher level features, such as the x and y coordinates of a detected face, could also be considered. However, these more complex features have a much higher computational complexity and would greatly reduce the rate at which images are processed. Since all of the features are calculated for each camera frame received, the feature with the highest complexity will be the limiting factor.

4. VISUAL OUTPUT

One goal of urMus is to provide an environment in which rich interactive media content can be written and designed. Part of that goal includes trying to find the right kind of programming language abstraction to make visual programming immediate and easy, yet as flexible as possible. This is in the spirit of visual programming and code as art as embodied in the design of Processing [16]. OpenFrameworks, a set of libraries in C++ has also been ported and used with iOS devices and is quite popular for mobile art projects [14]. The goal is to be much closer to the concept of Processing and other specialized environments for art programming by gearing not only the API but also the environment and language for the task at hand.

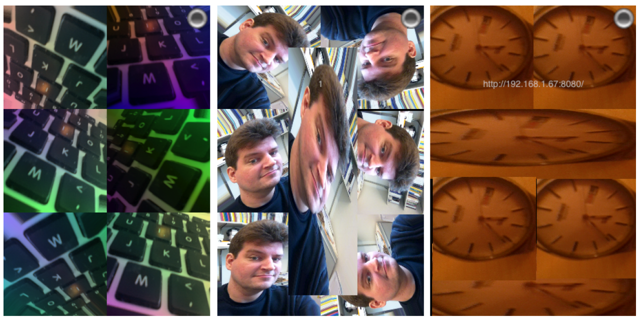

Figure 3: Examples of live camera feeds within multiple regions in urMus subject to a range of texture and color transformations.

urMus already comes with a rich and flexible graphical layout system that uses the concept of textures to create visually appealing details [9]. In order to allow visual programming, a texture in urMus acquires two functions. The first is that of a canvas. Graphical manipulation primitives can be applied to a texture to render into it. Currently urMus supports a set of graphical drawing primitives that is close to the set offered by Processing for 2D rendering. For the second function, textures also serve as brushes that can be used with any rendering primitive. This makes it easy to generate fairly complex visual content with simple primitives thanks to the use of complex and possibly changing textures. Furthermore, one can explore iterative and recursive painting ideas by repeatedly changing the texture roles of brush and canvas. Finally, as each texture can be flexibly moved, resized, and rotated on screen via the layout engine, one has versatile interactive control.

The camera image should be a flexible component of a visual mobile part piece. Current solutions often are inflexible. In most mobile situations, the camera input is just directly mapped to the full size of the screen. In urMus the camera image is directly fed into an OpenGL texture, which can be used in arbitrary number of instances and independently manipulated. This has a number of implications. For one all the standard texture and region manipulation capabilities of urMus do apply, such as tiling, rotating, stretching, and skewing that is possible by changes of texture coordinates and size of the containing region. Furthermore the camera texture can also be used as brush, hence one can actively draw and use all the 2D drawing primitives discussed earlier in this paper. This flexibility gives the artist many interesting ways to display what the camera "sees." At the same time the camera becomes part of the repertoire of visual information to create new content. These uses of the camera are fully interactive and multiple instances of camera images can be manipulated independently.

4.1 Rendering Primitives

The 2D rendering primitives of urMus can be roughly categorized into three groups. The first group consists of actual drawing functions which are `Point(x,y)`, `Line(x1,y1,x2,y2)`, `Rect(x,y,w,h)`, `Quad(x1,y1,x2,y2,x3,y3,x4,y4)` and `Ellipse(x,y,w,h)`. These allow for the display of points, lines, rectangles, arbitrary quadrangles, and ellipses. The second set of functions influence how these primitives are rendered: `SetBrushColor(r,g,b,a)` sets the brush color, `SetBrushSize(s)` changes with width of the brush, and `SetFill(b)` toggles whether or not the primitive is filled (if it is a closed primitive such as a rectangle or ellipse). The last set is texture control. If the command `UseAsBrush()` is invoked on a texture then future drawing and brush commands will use this texture as brush. This will continue until another texture is assigned as brush. All these operations are member functions of a texture, hence any texture can be drawn into, and any texture can be assigned to be the current brush. Finally, as any texture is by definition part of a region in urMus, it can be flexibly resized, positioned, layered and tiled.

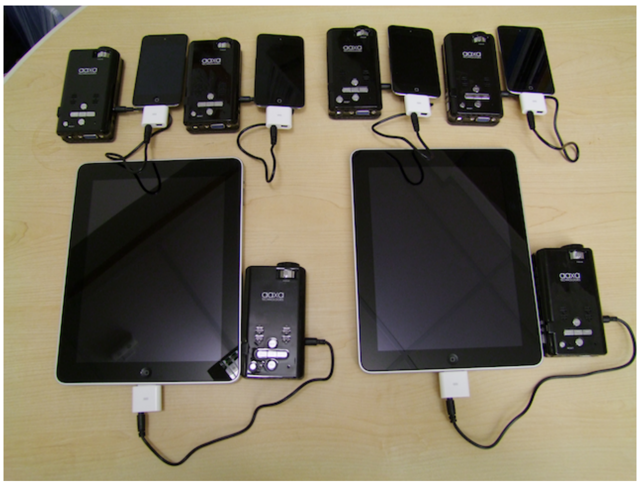

Figure 4: A ensemble setup of mobile projectors and their driving mobile devices.

Figure 2 shows the results of examples written in urMus. The leftmost example shows a tiled canvas drawing program. The canvas consists of four independent regions, which can be locked, unlocked and moved around on the screen. The painting will take the changed position into account, leading to the ability to continuously reiterate over the same image with different canvas arrangements, creating changing symmetries. The right example is the generic 2D rendering primitives text showing line, filled, and texture-based drawing of all basic primitives available. The thick-lined primitives are using a circular texture as brush and alpha-blending is active.

4.2 Output Technologies

Our work has looked at three different ways to extend the visual output capabilities of a mobile device for performance situations. The mobile multi-touch screen itself already serves as a rich and very useful display and larger portable devices such as the iPad extend the visual possibilities. Yet advances in mobile projector technologies are allowing to further expand and change the types of visual display that are possible.

This technology is still in its infancy but is already quite useful. We use Aaxa Technologies pico projectors which offer 33 lumens of intensity at a battery-time of roughly 30 minutes. This is too low for use in an ordinary lit room, but quite useful in rooms with dimmed lighting conditions. With this technology it becomes possible to tile multiple images, project on arbitrary surfaces and objects and create varied visual content while on the move (see Figure 5). The projectors can serve as much as a flash-light as a display. The mobile projectors are connected to the device using a video-to-dock connector and OpenGL content can be rendered onto external displays at interactive rates. Currently we are using ten such projectors (six of which can be seen in Figure 4).

The last form of output we have considered is the camera flash. The iPhone 4 contains a powerful LED flash and the feature is becoming standard on the latest mobile devices.

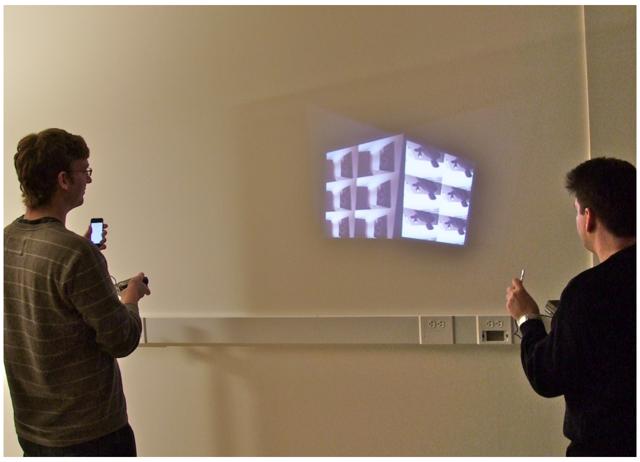

Figure 5: Two devices with pico projectors displaying a composite image based on their respective camera inputs.

The flash can be set to turn on and off at a variable rate which creates a stroboscopic effect.

4.3 Camera Integration

In the context of urMus, the graphical display is entirely controlled by OpenGL which has the benefit of being cross-platform. Also, it instantly gives the graphical versatility we desire. Once the camera input has been rendered to an OpenGL texture, any kind of transformation can be applied without effecting other instances. As mentioned above, the camera images are processed asynchronously on a secondary thread which is necessary to keep the user interface responsive. This presents a problem for the OpenGL pipeline because only one OpenGL context can exist on a thread at a time. To work around this, a new context is created on the secondary thread that uses the same sharegroup as the main thread's context. When two contexts are members of the same sharegroup, all texture, buffer, and framebuffer object resources are shared. When the very first frame of camera pixel data is received, a texture is created, the pixel data is copied into the texture with `glTexImage2D()`, and the main thread's context is made aware that it has access to a camera texture. All subsequent camera frames render into the texture using `glTexSubImage2D()` which redefines a contiguous subregion of an existing two-dimensional texture image. This eliminates the need to re-create textures with every new frame which saves computational costs and also prevents interference between the actual display of the texture on the main thread and the texture update process. Following this approach it is possible to retain interactive rates even if multiple copies of the camera texture are in use.

Access to camera texture is made possible through an extension of the texture API of urMus. By setting the `UseCamera` option of a texture instance, this texture will start using the current camera texture for all its texture-based operations. If the device offers multiple cameras (such as a front and a back-facing camera), these can be selected using the global `SetActiveCamera(cam)` API function. Currently all active camera textures are affected by this, as it current iOS devices do not allow multiple cameras to be active at the same time.

Currently iOS cameras operate at 30 frames per second and we found that multiple camera textures can be active while retaining interactive and that the performance is independent of the choice of camera. A test case with 30 active camera textures of various sizes gave a performance of 25 frames per second display update on an iPod Touch.

The lamp of the flash can be made to oscillate at a fixed rate using the `SetTorchFlashFrequency(freq)` global API. Due to technical limitations of current iOS devices this method is only available if the backfacing camera associated with the flash is used to drive camera textures.

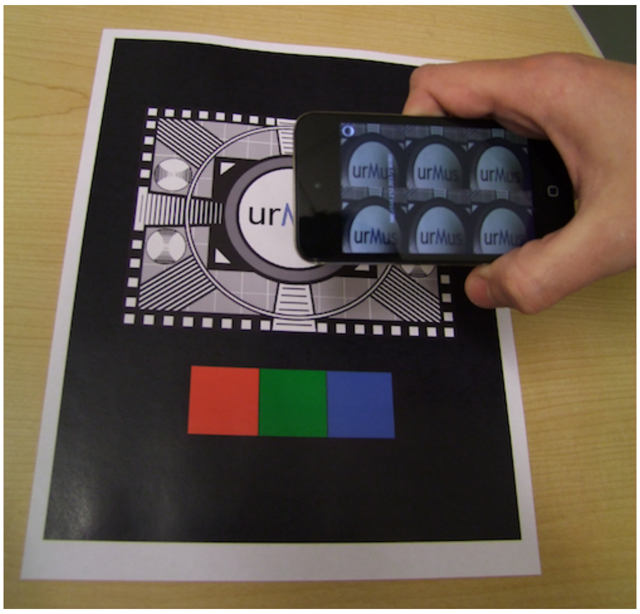

Figure 6: Features of the camera image are used to control audio.

4.4 External Display Integration

Since iOS 3.2 it is possible to be informed about an external display being plugged in and its resolution. One can then attach views that will be rendered on the external display. Currently in urMus, an external screen is automatically detected and the OpenGL rendering is redirected to the external display.

A test image with six live camera textures will render on an external display at 30 frames per second or above using an iPod Touch. This frame rate varies by less than 5 fps if the resolution of the external display is changed.

5. EXAMPLES

A vast area of performances can be imaged using the techniques discussed above, ranging from the use of the mobile device's display as visual augmentation to complex uses of camera input coupled with multi-media outcomes. Next we discuss but a few possible examples that we have implemented so far using urMus.

5.1 Performing the Image

"Performing the Image" is a visual performance that uses a prepared printed sheet with with color and textures to allow performance of sound over the image. Using the live-patching graphical interface of urMus, the performer can change the sonic realization of the image on the fly with simple multi-touch interactions by using features extracted from the camera signal as sources to drive synthesis patches. Color and edgy aspects of the camera image create a performantive canvas which can be explored by moving over different regions of the sheet (see Figure 6). This gives the piece a synesthetic quality by transforming the visual into the sonic.

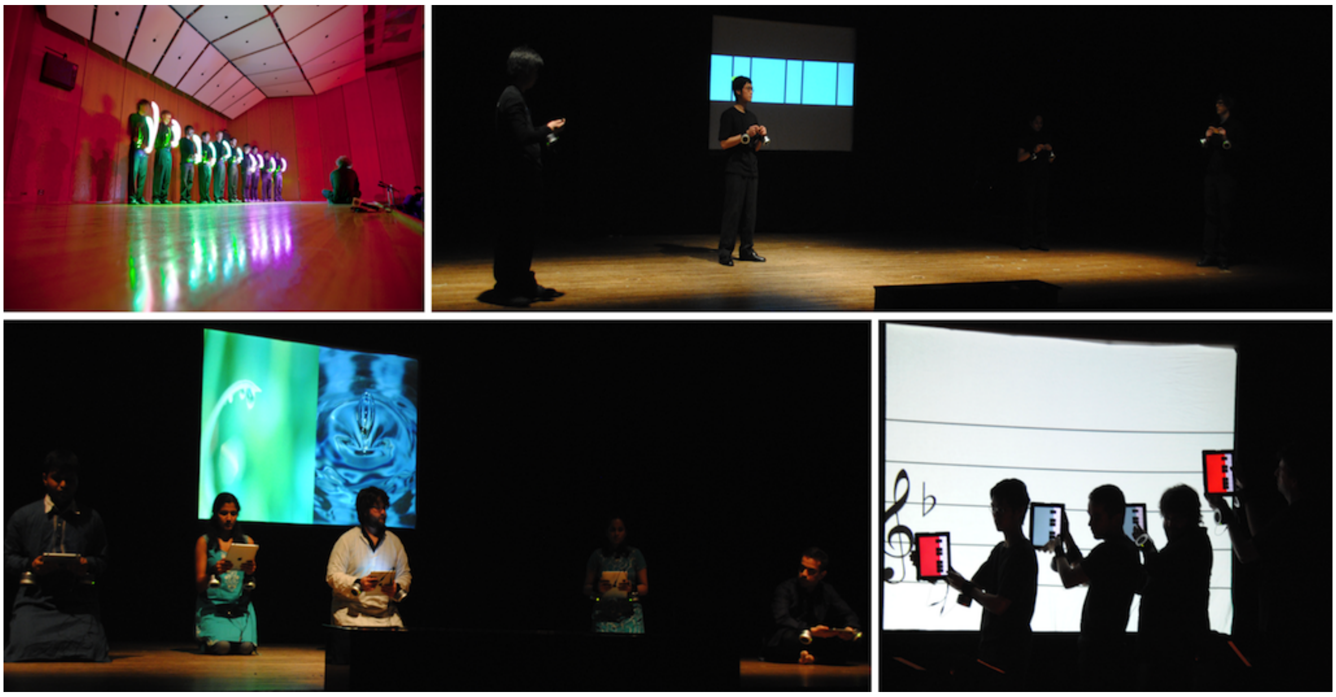

Figure 7: Examples of the Visual in pieces of the Michigan Mobile Phone Ensemble.

5.2 Visual in Mobile Phone Ensemble Performances

A key problem in mobile music performance is the explanation of the performance to the audience. There is no canonical understanding of what mobile music performance should be and very often visual communication is a big part of this explanatory task. A good example of this is the piece Color Organ, written by Kyle Kramer as part of a class taught at the University of Michigan on the topic of using urMus. As seen in the bottom right of Figure 7, four performers stand in front of a back-projected screen showing a musical staff. The performers hold the mobile device facing the audience. The screen of the mobile device itself is critical for explaining the piece to the audience. The performers lift the devices and place them in the correct position on the staff while colors express octave-matched notes.

Even static information can help strengthen the perception of a piece to the audience. In the piece JalGalBandi by Guerrero, Dattatrhi, Balasubramanian, and Jagadeesh uses visual projection to reinforce the sound. The piece transforms traditional Indian performance into an ecological perception of water and the visual display helps reinforce the kinds of water sounds that are currently creating the sonic experience (see bottom left of Figure 7).

Space Pong by Gayathri Balasubramanian and Lubin Tan uses networked communication to pass a virtual ball between performers. While gestures to symbolize that a ball is being passed around, the networked communication of the piece is not apparent. After all the transitions of sounds could have been due to actions of each individual performer and not some exchange. Here the projected visual display is also included in the network and depicts the interactions and changes that are induced by the performer's actions and it creates a visual appearance of an virtual ball moving in a virtual performance plane (see top right of Figure 7).

The importance of visual communication becomes even more critical if the devices used are small. In the Ballad of Roy G. Biv by Devin Kerr the screens of mobile devices are turned into mobile colored dot arrays. The piece is performed completely in the dark. Figure 7 shows a long exposure shot of the performance. Each color has a musical loop associated with it and the change of gestures in space create phasing effects and interplay that is intricately linked with the visual appearance of the piece (see top left of Figure 7).

6. CONCLUSIONS

In this paper we discussed a range of aspects regarding the visual in mobile music performance. Visual information can be used both as input and as output in musical performance. In order to make it easy for artists to create new mobile music performances with visual contributions, we have discussed how both visual input and output is facilitated within urMus, a mobile performance meta-environment. By combining camera capture with the generic OpenGL texture rendering engine, camera images are made flexible and objects of manipulation. Combined with textural rendering primitives, the camera can become a brush. For output, we discussed how textures can serve as both canvas and brush and therefore lead to a range of visual performance ideas such as rearranging canvases or recursive visual content. The emergence of mobile projectors extends and liberates the visual display, and multiple performers can join in creating content not just by what is shown, but also by where it is directed or moved.

Current technology is still limited by the computational power of the mobile device. While simple computer vision algorithms can be easily implemented, richer visual features are still too expensive to extract at interactive rates. Finding ever more complex sets of visual control and display remains a topic for future work as does the exploration of the vast possibilities of mobile display technologies in interactive mobile performance.

7. ACKNOWLEDGMENTS

This material is based upon work supported under a National Science Foundation Graduate Research Fellowship. Thanks to the student of the University of Michigan couse

"Mobile Phones as Musical Instruments" who suffered through the process of trying out the ever evolving immature early implementation of visual programming in urMus: Gayathri Balasubramania, Yuan Yuan Chen, Chandrika Dattathri, Alejandro Guerrero, Andrew Hayhurst, Kiran Jagadeesh, Steve Joslin, Kyle Kramer, Billy Lau, Michael Musick, Lubin Tan, Edgar Watson. This work was supported in part by a Curriculum Innovation Grant by the Office of Undergraduate Affairs of the College of Engineering of the University of Michigan.

8. REFERENCES

[1] S. Boring, D. Baur, A. Butz, S. Gustafson, and P. Baudisch. Touch projector: mobile interaction through video. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 2287–2296. ACM, 2010.

[2] N. J. Bryan, J. Herrera, J. Oh, and G. Wang. Momu: A mobile music toolkit. In *Proceedings of the International Conference for New Interfaces for Musical Expression*, Sydney, Australia, 2010.

[3] A. Camurri, B. Mazzarino, and G. Volpe. Analysis of expressive gesture: The eyesweb expressive gesture processing library. *Gesture-based communication in human-computer interaction*, pages 469–470, 2004.

[4] X. Cao and R. Balakrishnan. Interacting with dynamically defined information spaces using a handheld projector and a pen. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 225–234. ACM, 2006.

[5] X. Cao, C. Forlines, and R. Balakrishnan. Multi-user interaction using handheld projectors. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 43–52. ACM, 2007.

[6] G. Essl. SpeedDial: Rapid and On-The-Fly Mapping of Mobile Phone Instruments. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Pittsburgh, June 4-6 2009.

[7] G. Essl. UrMus – an environment for mobile instrument design and performance. In *Proceedings of the International Computer Music Conference (ICMC)*, Stony Brooks/New York, June 1-5 2010.

[8] G. Essl. UrSound – live patching of audio and multimedia using a multi-rate normed single-stream data-flow engine, 2010. Submitted to the International Computer Music Conference.

[9] G. Essl and A. Müller. Designing Mobile Musical Instruments and Environments with urMus. *Proceedings of the 2010 Conference on New Interfaces for Musical Expression*, pages 76–81, 2010.

[10] G. Essl and M. Rohs. Interactivity for Mobile Music Making. *Organised Sound*, 14(2):197–207, 2009.

[11] R. Gonzalez, R. Woods, and S. Eddins. *Digital image processing using MATLAB*, volume 624. Prentice Hall Upper Saddle River, NJ, 2004.

[12] A. Misra, G. Essl, and M. Rohs. Microphone as Sensor in Mobile Phone Performance. In *Proceedings of the 8th International Conference on New Interfaces for Musical Expression (NIME 2008)*, Genova, Italy, June 5-7 2008.

[13] K. C. Ng. Music via Motion: Transdomain Mapping of Motion and Sound for Interactive Performances. *Proceedings of the IEEE*, 92(4):645–655, Apr. 2004.

[14] OpenFrameworks. http://www.openframeworks.cc/.

[15] J. Park and M. Kim. Interactive display of image details using a camera-coupled mobile projector. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, pages 9–16. IEEE, 2010.

[16] C. Reas, B. Fry, and J. Maeda. *Processing: A Programming Handbook for Visual Designers and Artists*. The MIT Press, 2007.

[17] M. Rohs. Real-world interaction with camera-phones. In *2nd International Symposium on Ubiquitous Computing Systems (UCS)*, pages 39–48, Tokyo, Japan, Nov. 2004.

[18] M. Rohs. Marker-based interaction techniques for camera-phones. In *IUI 2005 Workshop on Multi-User and Ubiquitous User Interfaces (MU3I)*, January 2005.

[19] M. Rohs. Visual code widgets for marker-based interaction. In *IWSAWC'05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems – Workshops (ICDCS 2005 Workshops)*, pages 506–513, Columbus, Ohio, USA, June 2005.

[20] M. Rohs. Marker-based embodied interaction for handheld augmented reality games. *Journal of Virtual Reality and Broadcasting*, 4(5), Mar. 2007.

[21] M. Rohs and G. Essl. Which one is better? – information navigation techniques for spatially aware handheld displays. In *ICMI '06: Proceedings of the 8th International Conference on Multimodal Interfaces*, pages 100–107, Nov. 2006.

[22] M. Rohs and G. Essl. Camus 2: optical flow and collaboration in camera phone music performance. In *Proceedings of the 7th international conference on New interfaces for musical expression*, pages 160–163. ACM, 2007.

[23] M. Rohs, G. Essl, and M. Roth. CaMus: Live Music Performance using Camera Phones and Visual Grid Tracking. In *Proceedings of the 6th International Conference on New Instruments for Musical Expression (NIME)*, pages 31–36, June 2006.

[24] M. Rohs and P. Zweifel. A conceptual framework for camera phone-based interaction techniques. In H. W. Gellersen, R. Want, and A. Schmidt, editors, *Pervasive Computing: Third International Conference, PERVASIVE 2005*, pages 171–189, Munich, Germany, May 2005. LNCS 3468, Springer.

[25] G. Wang, G. Essl, J. Smith, S. Salazar, P. R. Cook, R. Hamilton, R. Fiebrink, J. Berger, D. Zhu, M. Ljungstrom, A. Berry, J. Wu, T. Kirk, E. Berger, and J. Segal. Smule = Sonic Media: An Intersection of the Mobile, Musical, and Social. In *Proceedings of the International Computer Music Conference*, Montreal, August 16-21 2009.

[26] G. Weinberg, A. Beck, and G. M. ZooZBeat: a Gesture-based Mobile Music Studio. In *Proceedings of International Conference on New Instruments for Music Expression (NIME)*, Pittsburgh, PA, 2009.

[27] M. Wilson, S. Robinson, D. Craggs, K. Brimble, and M. Jones. Pico-ing into the future of mobile projector phones. In *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, pages 3997–4002. ACM, 2010.