Networked Pixels: Strategies for Building Visual and Auditory Images with Distributed Independent Devices

Sam Ferguson
Creativity & Cognition Studios
Faculty of Engineering & IT
University of Technology Sydney
samuel.ferguson@uts.edu.au

Anthony Rowe
Squidsoup.org
ant@squidsoup.org

Oliver Bown
Interactive Media Laboratory
UNSW Art and Design
o.bown@unsw.edu.au

Liam Birtles
Bournemouth University
lbirtles@bournemouth.ac.uk

Chris Bennewith
School of Art, Design & Architecture
Plymouth University, UK
chris.bennewith@plymouth.ac.uk

ABSTRACT

This paper describes the development of the hardware and software for *Bloom*, a light installation installed at Kew Gardens, London in December of 2016. The system is made up of a set of nearly 1000 distributed pixel devices each with LEDs, GPS sensor, and sound hardware, networked together with WiFi to form a display system. Media design for this system required consideration of the distributed nature of the devices. We outline the software and hardware designed for this system, and describe two approaches to the software and media design, one whereby we employ the distributed devices themselves for computation purposes (the approach we ultimately selected), and another whereby the devices are controlled from a central server that is performing most of the computation necessary. We then review these approaches and outline possibilities for future research.

ACM Classification Keywords

H.5.5 Sound and Music Computing: Systems; I.3.3 Picture/Image Generation: Display Algorithms

Author Keywords

Distributed, Pixel, Display System, Internet of Things, Audio

INTRODUCTION

Many examples exist where the notion of 'distributed computing' has been explored as an artistic medium. However, with the rise of Internet-of-Things type devices, new configurations of media now exist where this notion takes a central role. In most cases installation media have included sensors which allow interaction of various types, and in many cases the sensors and display systems have been built into site-specific systems. With the rise in capabilities of internet of things (IoT) type electronic devices, and their commensurate decline in cost, more options are now available for the development of systems that distribute their computing power and display technology, rather than centralising them at a central computing unit. Similarly, sensor technology is now quite capable of being dealt with as a set or flock of data rather than as single inputs processed at a central hub. The affordances of IoT technology present media artists with the potential to make scaleable site-specific artworks with custom hardware, all in rapid time and at low cost. However, there are many risks involved in this process, as well as creative organisational challenges, and at the same time a number of strong design principles that we can draw on. At this stage there is little discussion of the creative and technical practice of working with IoT-based media artworks.

Figure 1. ***Bloom*** **installation installed at Kew Gardens, London. Image Copyright Anthony Rowe 2016.**

CC '17, June 27-30, 2017, Singapore, Singapore

ISBN 978-1-4503-4403-6/17/06…$15.00

DOI: http://dx.doi.org/10.1145/3059454.3059480

In this paper we will focus on our experience of developing the hardware and software systems for *BLOOM*, an installation developed for Christmas at Kew, a light show held in Kew Gardens, London in 2016. The hardware and software developed for this system were bespoke, and the work represents a significant shift in the architectural underpinnings for this type of artefact.

Background

Systems of media display that involve screens made up of pixels are the central display component of the computing systems that have dominated the last century's advances in technology. As computing systems become ever smaller, and costs are driven down, the possibilities for new types of display system have expanded, and research interest is beginning to focus on the characteristics and design methods for displays made up of distributed pixels.

'Media multiplicities' are an emerging category of artistic and designerly practice [4]. Media multiplicities are made up of groups of independent elements which function together to act as an artefact of their own but which can usually also act as a substrate for other content to be displayed. They may be described generally by 3 main axes:

heterogeneity vs. homogeneity: whether the multiplicity is made up of heterogenous or homogenous elements, content or positioning.

object vs. substrate: whether the multiplicity is primarily understood or experienced as a group of objects, or as a substrate for other content to be displayed

composed vs. self-organized: whether the multiplicity is centrally controlled or composed, or self-organised based on rules.

This method of description helps us to understand not only how to design multiplicities, but also how to extend the design to achieve different goals.

Ishii has discussed the relationship between displays, interfaces and information in the process of defining tangible interaction [15, 14]. By moving interaction from a command-line, to a graphical, to now a tangible user interface, users can exploit the similarity between skills they use to interact with their physical environment, and skills they can use to interact with computing systems with these new interfaces. In such a context, computer systems are inevitably tending towards becoming distributed, rather than centralised, and systems that can augment and control these distributed systems will be a critical component of the technological future.

Research in Distributed Displays

Research interest in distributed arrays of pixel-like devices has resulted in several systems being developed. Seitinger et al. [26] developed a system of 50 'urban pixels' which they used to explore concepts relating to the potential of distributed displays. They identify flexible placement, autonomous power, unbounded placement, variable resolution and responsivity as important characteristics of these new distributed systems. Their system employed interaction in two ways - the pixels could receive 1-character SMSs from the audience to allow them to be controlled pixel-by-pixel, or they could be 'painted' by using a flashlight to activate pixels in brush-like fashion. Barker et al. [1] and Hauesler et al. [11] focused on the development process and capabilities of firstly 'Janus' (a face-like facade of addressable pixels), and then 'polymedia pixels', a system of networked pixels. *Squidsoup* has a long history of developing volumetric light installations made up of thousands of individually addressable LEDs, positioned in a 3-dimensional grid [21, 20]. Sato [22, 23] has described how a set of small independent and distributed pixels would be affixed to a non-regular surface, and then by using a camera the location of each pixel would be known with high accuracy, allowing the precise control necessary to present graphical images. Fischer et al. [10] designed a system of kickable and flickable light shards that used irregular shape to modify their physical behaviour. *Junkyard Jumbotron* [2] showed it was possible to flexibly employ groups of heterogenous mobile devices to display images that spanned multiple screens with a simple architecture, as did Schmitz et al. [24]. Schwarz et al. [25] followed this work by simplifying the requirements for using a 'Phone as a Pixel' even further.

'PushPins' [7] were small computing devices that could be powered by pushing their contacts into a foam that included layers of conductive and non-conductive material. Each computer could be repositioned easily across the power substrate, and when they were powered they built their own network of devices local to each other. They were programmed through an Infrared 'flashlight' that beamed software through an infrared interface to each of the 100 nodes. Paradiso [18] went on to discuss *Tribble*, a system of tiled computers that formed a sensate sphere, but which had no central controller, as well as *Z-Tiles*, a system of tiles for a sensate floor. Bown et al. [5] described Distributed Interactive Audio Devices – distributed systems for interaction with audio synthesis capabilities, designed to be physically handled and passed between participants that are part of an audience, and which can be used to develop simple musical games [3].

The effects of the motility of distributed pixels is also an area of emerging research. Kuznetsov et al. [16] developed Wallbots, which were robots that could traverse metal walls with their magnetic wheels, but which also had display elements and light sensors for interaction. Similarly, *Spaxels*, a set of first 16, then 100 quadrotor drones outfitted with RGB LEDs, demonstrated the capability to coordinate their movements so as to produce images and animation [12]. Digurmati et al. [9] have extended this work to allow for image animation with their *Pixelbots* system of small mobile RGB pixels. On a much smaller scale, Dantu et al. [8] developed methods for programming small sets of tiny programmable flying devices outfitted with sensor capabilities. Merrill's *Siftables* are computers within tiles that implemented interaction systems employing physical rearrangement interaction procedures (like scrabble or dominos) as input for the devices [17]. While not independently mobile, they used careful sensing of their physical position as a prime source of interaction input in their applications. Vallgarda et al. developed the notion of 'programming materiality' [27], by citing techniques such as *tangible programming* and *programming by example*, resulting

in a set of speculative tools that can be used to program physical interfaces. Wiberg, [28] and Robles and Wiberg [19] have investigated notions of materiality in interaction design. These works show that as groups of devices become more advanced, and are used for more complex purposes, advanced continuous material and positional sensing may need to be integrated with the display algorithm to allow elements to move independently of the image or content being presented.

Aims and Overview

The aforementioned examples have generally been small-scale demonstration deployments, with a lot of the systems being between 2 and 100 units. At this scale many critical challenges of code deployment and content generation are not as significant, as it is generally quite straightforward to transfer compiled code with a simple script. Furthermore, even small scale network infrastructure are mostly all well-suited to managing even this number of connected devices. Few of these systems focused on developing other modalities either – displaying sound in addition to visual media was generally ignored, or at least have not been carefully explored. Many of the devices discussed were developed with the engineering challenges and system capabilities being the focus of the reporting, with little research discussing how a practical workflow for creative design could be employed by artists and media designers working with these display systems.

In the work we describe below there are several important advances from the aforementioned systems.

- this is the largest deployment of distributed devices we know of, our system included 923 devices in its initial deployment, with a remaining 77 to be added when they are reworked.
- each of our devices included the capability to produce sound, and designing this audio content became a critical part of the design workflow.
- this system used a GPS sensor on each device for individually locating each pixel.
- Finally, this system required the ability to be installed for a period of 6 weeks in an inhospitable environment, and to function both reliably and in an aesthetically pleasing manner to fulfil a design brief.

The following sections will describe first a description of the hardware and software used in the system, including discussion of the design decisions made both during the hardware design, and their effect on its capabilities. Secondly, we will move to a description of two alternative approaches to developing digital media for this system, alongside the workflow for media design and deployment. Finally, we focus on methods and directions to improve the system capabilities for the future.

SYSTEM DESCRIPTION AND DEVELOPMENT

The devices used in this installation were built from a number of individual elements brought together as an integrated system. The constraints on the design included a minimum number of elements (1000) required by the client, alongside a budgetary constraint, combined with the desire to extend the capabilities and reuse the system in the future.

Figure 2. The devices and their cabling were required to be waterproof and durable, given the installation location and conditions and the 6-week installation period. The stands were made from flexible plastic, and carried the DC 12V power supply that powers each device. Image Copyright Anthony Rowe 2016

Installation Design Context

The design context for the installation was quite specific. The open space of the Kew Gardens site allowed us to employ GPS tracking without the occlusions of an indoor or urban setting, and also worked to allow the large scale of the installation to be fully appreciated.

The durability and reliability of the installation was critical as the system was to be deployed for a six-week installation. The system was to function without supervision and without significant monitoring, except for a daily initiation phase. No non-automatic debugging could take place after the launch of system. Weather conditions were harsh at that time of year in the UK (see Figure 2 for a general impression) – moisture and extreme cold were expected to be issues, and so watertightness was a critical issue for the devices. The location was well-suited to the use of a wireless network, as there was no network infrastructure in place to interfere, and there were few occlusions given the level surface. However, the practicality of such a large number of devices being attached to a single wi-Fi network was a significant complication. Furthermore, the site was bisected by path across which no cables could be run.

Device

The device was designed to be lightweight, low-power, network enabled and to create sound and light. It included several important capabilities:

Onboard Processing: The device used an ESP3266 type module (a Digistump Acorn) which included an over-the-air programmer and various other advanced capabilities.

Onboard WiFi Networking: One of the capabilities provided by the Acorn module was onboard wireless networking.

GPS: a compact GPS aerial and chipset was soldered onto the circuitboard and connected to the Acorn module directly.

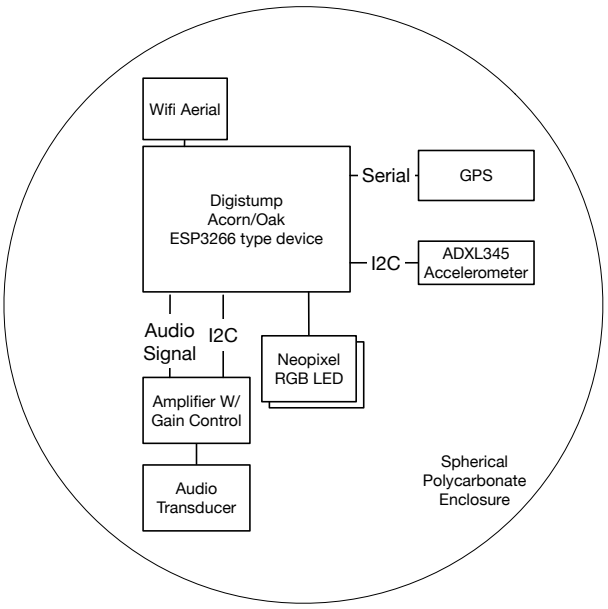

Figure 3. Diagram of components of the distributed pixels.

Accelerometer: An ADXL345 accelerometer was included on the device, and connected to the Acorn module by an I^2C interface.

Audio amplifier: a simple audio signal amplifier was included whose gain was digitally controllable with high temporal resolution, through an I^2C interface.

Transducer: a contact-type transducer was glued to the device, so that sound produced resonated through the device housing.

LEDs: Two NeoPixel type RGB LEDs were positioned on either side of the device.

The device was designed to be designed to be affixed to the internal base of a polycarbonate globe, and included contacts for the incoming power cable to be soldered to the centre of the device. It was circular in shape and LEDs were placed on either side of the circle in order to obtain uniform light distribution. Figures 3 and 4 demonstrate the components of each pixel.

Sound

As far as we are aware, while many systems of distributed devices have existed previously, most have focused on visual display, and few have included sound producing hardware on the number of elements we are dealing with. Our device is one of the first to include audio hardware as part of the design. The audio electronics on the device were rudimentary as the ESP3266 chip is not well-suited to audio signal playback (it splits processor time between multiple tasks) and the main sounds we could produce were simple tones using the `tone()` command. However, we were able to include an amplifier with I^2C gain control that could be controlled with high levels of temporal accuracy. Having fine-grained control over the audio amplitude allowed simple amplitude envelopes as well

Figure 4. Components of distributed pixel system with spherical enclosure covers removed. The GPS unit is clearly visible, as are the 2 NeoPixel LEDs and the audio transducer. The audio transducer produced sound by transmitting vibrations to the circuit-board and through to the spherical cover, which was a much more space efficient approach than trying to include a loudspeaker and associated mounting would have been. Because of the size of the elements, the circuitboards were able to be mounted on the lower-most portion of the sphere, meaning that the rest of the sphere was not in shadow. Image Copyright Sam Ferguson 2016.

as longer-term ramping effects, meaning tones that are sometimes annoying when they are constant, could be modulated both over the term of a note (200-1000 ms) and over the term of a *crescendo* or phrase (10-60 s).

To program the sound we took a tightly synchronised approach, whereby both the brightness of the LEDs and the audio gain were usually linked in some manner. Usually, each visual pulse was accompanied by an auditory pulse of the same length and temporal shape, but with an independently chosen pitch. This meant that visual textures were accompanied by matching auditory textures, which simplified the design and programming of the devices. This also streamlined the experience for participants as the visual and auditory stimuli could be fused, and it was easier to work out which pixel was making a particular sound. For the purposes of experiencing the installation as an embodied space, this tight coupling was a way to rapidly perceive the temporal and spatial distribution of the media. Nevertheless, there was no hardware requirement that this approach be taken – future designs may employ differing

approaches to the relationship between the auditory and visual medium.

Server and Development Cycle

In most development contexts, development iteration takes place when the code is compiled (or interpreted) and then executed, which shows whether the program performs as expected, resulting in modification and an iteration loop. For distributed devices, however, where the number of devices exceeds a practical deployment limit, taking the simple approach of compiling, using a serial port to upload the firmware over a cable, and then and executing the code on each device was obviously prohibitively slow. Furthermore, in many situations we found testing with a single or a couple of devices was not capable of uncovering critical bugs that were intermittent, but were immediately found when the program was deployed with the planned number of devices. Finally, to demonstrate and adjust the planned aesthetic effects we needed to see all the devices in use. We therefore could not rely on the typical iterative design approach to programming, and a different approach was necessary.

Digistump provided a lightweight *node.js* server which ran on the development machine, for distributing the firmware images to each of the devices. The server was queried by each device when they booted, and periodically when they were running. If the MD5 hash of the current firmware image on the server was different to that on the device then an upload process was triggered, and the device replaced its current firmware with that found on the server. The server was capable of serving the firmware image to all 1000 devices rapidly and almost simultaneously, meaning we could upload a new firmware image approximately 90 to 120 seconds.

This server was partnered to an Arduino extension designed for working with this device. It had the capability of being able to upload an image across a USB-to-serial adapter, or alternatively, compiling a firmware image directly to a folder monitored by the aforementioned *node.js* server, which would trigger the update to occur. Firmware image updates using the server could be both seamless and rapid, and differed little from the process of uploading an image via USB, except that the number of devices that could simultaneously receive updates was almost unlimited. Despite its rudimentary nature, working with Arduino as one of the development environments was familiar for us as we possess experience as creative coders, and this was a strong advantage over more technical integrated development environments. Depending on the programming platform, other approaches exist that achieve similar aims, for instance [6] which allows the serialisation of java classes and their immediate execution by a running device without being rebooted.

While other deployment systems exist that can deal with large sets of devices, especially at the enterprise scale, they usually involve significant complexity, configuration and cost and are rarely designed for creative purposes. With this somewhat more lightweight approach we found a solution that suited the scale of our creative development process but allowed us to treat the large scale of our device set similarly to a small scale system.

In terms of network configuration, to provide enough connectivity we used a set of 4 separate Xirrus Networks WiFi Access Points connected to a central switch by PoE ethernet cables. These access points were designed to be capable of maintaining hundreds of simultaneous connections. Two of the Access points were placed on one side of the bisecting path, and the other two were placed on the other side, connected to each other by a microwave link.

Graphical User Interface

To support the development process and to allow rapid deployment we developed a partner graphical user interface for issuing commands to the system. It used a graphical layout to allow a user to locate each pixel based on its position as reported by each device's GPS data. It also simplified the communication with the devices as each of the main set of OSC commands used to control the multiplicity was rendered as a button on the interface.

For some situations we were able to use a set of sliders and selections provided by the interface to program new modes directly. This meant that we could experiment, during deployment, with particular parameter settings in the actual installation location. However, as the development process proceeded, and the required behaviours became more complex, using our simple GUI interface was not flexible enough to describe many of the more complex options. We therefore spent much of the development time coding these parameter selections manually into the GUI framework and triggered each behaviour using particular buttons.

Finally, as the individual behaviours were completed, we then programmed them into a sequencer which could be controlled from the interface. The system was deployed using this sequencer to synchronise the sending of each of the mode instructions.

APPROACHES TO DISTRIBUTED VISUAL IMAGERY

During the development process we noted two alternative approaches to developing digital media for distributed pixels. There was one approach whereby the devices were programmed to behave in certain ways in response to a global command, based on their location and on other data that was local to the device. We termed this the Computation-on-Device approach. The alternative approach was to obtain data pertaining to each device at a central control point, fuse it together, and then broadcast much simpler commands to each device in this collective display, the overall image could be controlled. We have termed this the Computation-on-Server approach. See Figure 5 for a diagrammatic representation. Of these two approaches, the one we implemented within the installation was the Computation-on-Device approach primarily for both design and implementation reasons, although we had a separate code-base containing an implementation of the Computation-on-Server approach which we prepared to cope with contingencies (although it has not yet been tested with the system).

Computation-on-Device

The Computation-on-Device approach is made up of several constraints.

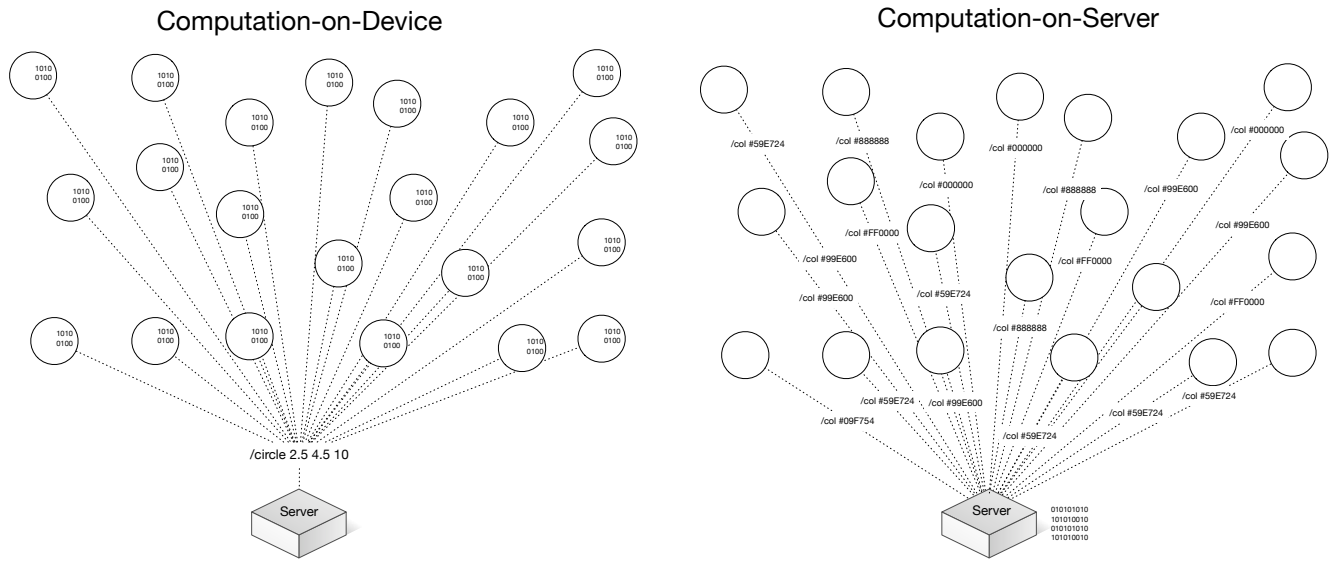

Figure 5. Two approaches to controlling distributed pixel display systems: Computation-on-Device and Computation-on-Server.

- identical commands are broadcast to every device;
- commands are often sent seconds, or multiple seconds apart;
- each device can use both parameters passed in the command, and data that is internal to the device ;
- the dimension of time is usually controlled independently by each device, at least on the micro scale;
- any animation effects are controlled by the device.

The decentralised approach allows for a great deal of redundancy with regard to network communication. Given the large number of devices we were attempting to control from the network, we were unsure of the reliability of sending frame-based network messages (as required by the Computation-on-Server approach). Notably, we will likely remain unsure about this given the differing network and environmental conditions in which the system will be deployed in the future, but it's also possible that new network software or different network configurations may help to make this less of a priority going forward.

In the Computation-on-Device configuration, the devices store the GPS location locally, and we design algorithms that command the devices to exploit this data within the context of a global command. An example is the simple expanding circle. This can be created by:

- sending a GPS location to each of the devices;
- using that location, alongside the locally stored location, to calculate a distance;
- using the distance to calculate a delay time, that is used to control the timing of the LED being activated.

While extremely simple, this algorithm can be quite effective when used on a large scale. Variations can achieve various different effects of course, in rather a similar way to typical vector graphical algorithms.

Figure 6. Installation as viewed from above. The work was installed in a space approximately 80m by 25m, with a path bisecting two halves of the installation. For context, the structure in the background is the Kew Gardens Palm House. Image Copyright Anthony Rowe 2016.

Computation-on-Server

The Computation-on-Server approach relaxes many of these constraints and is generally reliant on both a fast network, and a centralised controller computer.

- individual media commands are sent to each device based on their location;
- media commands are sent once every frame - at a typical frame-rate of between 16 and 60 frames per second;
- usually any sensor data obtained by each device is sent to the controller machine, rather than being stored locally;
- time is controlled by the central controller;
- and therefore animation is also controlled by the central controller.

This approach theoretically gives a lot more flexibility regarding media content – there is no need to determine shapes through algorithms that respond to location data. Instead the locations of each pixel are determined in advance and the media content is forwarded to each of them individually. Many images and patterns are not straightforwardly described by geometry or algorithms, and therefore using a Computation-on-Device approach can limit the types of content that can be forwarded globally. Furthermore, each image or pattern needs to be constructed algorithmically – a significant limitation for a creative process.

The main drawback of this approach is that an individual message must be sent to each pixel who's colour needs to change, once for every frame. For 1000 pixels at 24 FPS, this can require 24000 individual messages per second, requiring high throughput on a network that is likely to be wireless and is therefore often working to capacity to maintain the many connections when numbers increase past a typical lower limit for a router. Furthermore, IoT devices that maintain wireless connections such as the Digistump Acorn often have to split their processing power between subsystem processes that maintain the connection and foreground processes that perform sensing and display. Increasing the network burden can also increase the likelihood of unexpected behaviour when the foreground process is performing particular functions. While many solutions exist to improve reliability and stability of large networks of this nature, as numbers of elements scale past 1000 pixels to even larger numbers this may continue to be an issue. Other systems have, for instance, used very low-bandwidth, high-availability network solutions (see for instance Sato [22] who used a wireless solution with a throughput of 4800 baud).

Comparing Approaches

These two approaches, Computation-on-Server and Computation-on-Device, are not mutually exclusive, and as design methods become more advanced it is inevitable elements of both approaches will be used. More advanced software and hardware will also play a role in merging these two approaches, and fusing the data the devices produce in smarter ways. In Table 1 we detail each of the different benefits and drawbacks of the two approaches.

In the case of the installation *Bloom* we opted primarily for the Computation-on-Device approach, but this was only made possible by the rapid deployment architecture that Digistump developed. Without this deployment architecture we would have necessarily needed to adopt the Computation-on-Server approach, as otherwise deployment time would have been prohibitive to changes. We considered many ways of implementing aspects that used Computation-on-Server, which will ultimately require more research, and possible greater sensor accuracy or fusion methods.

A drawback of both approaches, but primarily the Computation-On-Server approach, is that by the very nature of distributed systems the images produced lack resolution. A distributed system (of either type) is unlikely to include elements that are either physically adjacent to each other, or positioned in a grid. Therefore, the typical approaches to improvement of raster images, such as anti-aliasing, or even more complex methods designed for pixel-art such as Inglis' recent work [13], will be confounded. More fundamentally though, if the gaps between pixels were to be filled by more pixels, it would raise the question of why each pixel is not physically wired to each of the other pixels, meaning they would not be distributed any longer. At its core, it seems that a distributed system of pixels engenders some level of pixel sparsity, meaning that graphical images of a bitmap type are not likely to be approached in the same way when displayed on a distributed system, without some fundamental new configuration of the way distributed pixels form an image surface.

Finally, location data is critical for either approach to display shapes and images accurately. Without highly accurate location data, pixels expected to be in one location can be in another, resulting in transmitted content being heavily distorted. While this is more apparent with images of a bitmap type, which may be more likely with the Computation-on-Server approach, adopting Computation-on-Device does not solve this problem. Shapes generated by algorithms computed on each device but which rely on localisation data can still be distorted, although they are less likely to emphasise the figurative nature of the media. Our choice of GPS for this technology was successful in practical terms, in that the principles were clearly capable of being tested, but more accuracy is definitely required to enhance capabilities in this respect. More modern localisation alternatives may theoretically yield more accurate information, although we were currently unable to identify cheap, well-implemented, independent, robust alternative solutions, that didn't require extra hardware beacons, which would scale to 1000 devices, and which could work within a large open parkland space, its likely that such solutions will become accessible in the near-term. Given that the devices are installed for long periods, one solution under investigation is long-term averaging of the GPS data, followed by the storage of that data either at an online source or on-board.

DISCUSSION

This article has discussed the development of a distributed display system. This is one of the largest distributed pixel installations we know of, and is one of the first to approach screen-like resolutions. It is also one of the first to include sound as an element of the distributed pixel system and to design distributed media that includes sound content as an element. It is one of the first to use individual GPS chips as a location sensor for each pixel and to employ that data as part of a visualisation algorithm. The new attributes and capabilities for this distributed system of pixels has opened up a lot of questions and possibilities that have necessitated further research.

In the preceding sections, several approaches to development have been discussed, and particular strengths of this system have been highlighted. With the Computation-on-Device approach global commands can be sent, and then patterns and simple geometry can be displayed based on the information local to the device interpreting the command. With a Computation-on-Server approach, individual messages are sent to each device separately. The discussed approaches describe new possible capabilities, but also hint at future chal-

Table 1. Comparison between digital media design based on Computation-on-Device and Computation-on-Server. Bold is used to highlight clear advantages of each of the approaches to the architecture of the design.

Area	CoD	CoS	Design considerations
Bandwidth	**Low bandwidth. Potential to broadcast instead of unicast the network data.**	High bandwidth. Must unicast messages.	Know your system's network capabilities given your expectations for framerate and number of devices (for the current project, and bearing in mind you may want to scale up the same piece in the future - also bearing in mind that improved hardware may cater for such scaling up). CoS may simply not be an option.
Resilience to network failure	**The system can keep behaving to some extent without a network. If there is partial network loss then the system needs to know this in order to respond coherently. It depends on the work whether this network-free mode is acceptable, and for how long.**	If some or all of the network goes down the system is paralysed / broken. In theory you could still have a fallback behaviour using CoD, but if you're going to have to create this (and can) then you might as well just take the CoD route.	CoD may be more 'aesthetically' robust to network failure. Questions include how likely network failure is and whether a networkless fallback option is actually acceptable and for how long (and if so, can it be trivial, such as random flashing lights, or does it need to be non-trivial). If the network is unreliable and a non-trivial fallback is acceptable (but a trivial one is not), then a CoD approach may be preferred.
Network performance worst-case scenarios	**Very likely to be no network limitations except under certain conditions where spikes in data might occur or high-bandwidth data is being sent.**	The network will have a fixed capacity for framerate / number of devices, which can be known from prior tests. If you're within that capacity the the only limit is the computational limit of the server.	If your system does have the capacity to robustly do CoS at the framerate you desire, then you can be sure it will not have spikes, as the device-side behaviour and amount of data being sent will be very predictable and robust. Still, your bandwith needs will be much less with CoD.
Network message failure	Critical that message is received, but being low bandwidth it is easy to send multiple copies of each message.	Occasional messages can be skipped without too much interruption, and if a pixel doesn't change it doesn't need to be sent.	There may be cases where there are aesthetic factors involved in your approach to failure, and either option may be advantageous. You may like it that the odd rogue device is off-script, and you may consider the network uncertainty to be a conceptual part of the piece. Assuming otherwise, then CoD may be more reliable, if there are doubts about network performance for CoS. Rough video compression techniques can be used to save sending data in CoS mode.
Development and debug experience	Device code in constant development. Having to co-develop and keep in synch two bodies of code. Last minute on-device debugging may be needed.	**Set and forget. Device code rarely needs updating and can be made very robust.**	Coding for devices such as Arduino can be much harder due to the more low-level nature of C/C++ and the fact you are having to run the code on a different device from the device you are writing the code on. Debugging support tools for writing and running code on a full OS-based machine are much more sophisticated. Also this means that the device code is ever changing as the project develops. This can put pressure on at critical deployment times where there may be a need to fix bugs and re-deploy code to multiple devices, although we found this deployment process to be relatively robust and painless.
Simulating device behaviour without access to devices	To simulate the device behaviour on server means to re-implement the device code on the server, although it is also possible to have an emulator that runs the device code in simulation.	**Device behaviour is easily simulated and can be simulated in development time. Since behaviour is on server it is trivial to 'display' it.**	Consider how important simulation will be to achieving your goal. Can you test on the real hardware? Is the final outcome likely to be unexpected or hard to imagine? Are you taking a creative iterate and test approach where you want to make creative tweaks as you go, or do you have a clear design in mind? Alternatively, perhaps the site-specific complexity of the work is such that even a simulation is unlikely to give you a good idea of the results (as in the case with multi-speaker works).
Establishing network communication 'vocabulary'	Depending on the type of data being communicated (assuming something does need to be communicated), may need large and constantly changing vocabulary of OSC (or other network) commands. Potentially more brittle.	**Very simple and relatively fixed vocabulary of OSC commands.**	Provokes consideration of design patterns: (1) extreme A - little or no network communication needed, devices doing their own thing, possibly using global clock / script (then use CoD), (2) extreme B - devices are rendering complex image that cannot easily be described in simple, low-computation terms (then must use CoS), (3) in the middle - devices have an enumerated series of modes (= code subroutines) that are chosen from top down, each mode may have various parameters, and each device may incorporate local data into its behaviour (accelerometers, GPS, ID), (4) building on 3, the device behaviours are not just subject to parametric control (numbers, enums, patterns, etc.) but more complex data that might be understood as "programming". This ignores issues of user interaction and inter-node communication. See below.
Suitability to parallel computing	Better for parallel style computations such as cellular automata, but worse for systems with top-down organisation.	Equally good for different sorts of computation, but computational power does not scale with number of devices.	Is your code easily parallelisable? e.g., a Game of Life CA model is an obvious example where the code is parallelisable - each device simply has to run its update rule based on the state of the neighbouring devices. The catch is that the devices need to maintain a list of their neighbours and send those neighbours their states at each time step (or listen out for states broadcast from neighbours), so that the neighbours can update. In this case even though the system is naturally parallelisable, it actually involves more data communication to run it using CoD than it would using CoS. This could be considered something of a paradox. By contrast, if instead of CA update rules there was a complex deep neural network calculating each next state, then this might be too much for the server to handle, necessitating a distributed approach.
Implementing device interaction	Interaction between devices needs to be mediated by server, hence potential for doubling up code. e.g., all devices need to maintain a model of where all other devices are.	Interaction between devices needs to be mediated by server, so natural to compute on server.	Some situations are inelegant and should be avoided: (i) a cumbersome and hard-to-maintain list of network communicated instructions that both device and server must know about, (ii) situation where the state of the device needs to be known on the server, or in other devices, and cumbersome messaging or 'modelling' is needed to keep the states in synch in different places. In these cases CoS can resolve the problem.
Debugging ease and bottlenecks	More debugging load needs to go on (a) the devices and (b) the correct network messaging.	**More debugging is on the server model. This is better supported in most IDEs.**	This is a major advantage for CoS in terms of the extensibility, maintainability, portability and clarity of program design. The network boundary has been simplified to something that is extremely easy to visualise.
Attitude towards creative constraints	Can be a good 'creative constraint' forcing a more creative modular design / VERSUS / it is unnecessarily constraining.	Can be too open-ended. "Creative coder's dilemma" / VERSUS / it is nice and open-ended. Freedom to choose your own constraints.	If the algorithmic description of the modes is very easy and contained, then this tends towards CoD. It is often appealing to set out with this objective.
Creative coding philosophy prompted by architecture	Encourages a distributed communication philosophy (objects not substrates).	Encourages more of a media-based / image-rendering-based philosophy (substrates not objects).	If the concept is not already preconceived then the choice of CoD versus CoS may possibly directed more by a motivation to engage with either one of these approaches. Particularly, the creative nature of a CoD architecture may offer a refreshing relief from the single-computer context, even if at a more fundamental level, given the linkage / boundarylessness provided by network communication, CoD and CoS may be more or less equivalent.

lenges with distributed systems, some of which we discuss below.

The creativity support tools that exist for distributed display systems are currently quite limited, and their research is not an area given much emphasis, probably because many of the systems we describe are generally at an early stage of development, or are demonstration systems only. Much of the media development takes place during the installation of a system. Tools and techniques are ad-hoc and are generally subordinated to the engineering challenges faced getting the system to function. Rather than being approached with generalised tools in a way that allows domain specialists to focus on this element, content creation is generally approached by the same people tasked with engineering the system itself. Its possible that new tools for content creation would probably involve several elements:

Simulation: Being able to simulate the distributed system Will allow designers and developers to rapidly iterate their design before it is installed.

Visualization and Sonification: Part of the ability to simulate a design for a distributed system is to visualise it and simulate the auditory experience. Also, being able to experience the visual and auditory impression from the expected traversal of the space may allow necessary adjustments to be implemented before installation.

Manipulation: Controlling parameter choices, manipulating positioning or density, and changing the aesthetic choices described by the code in an intuitive fashion is an approach that is common across almost all fields of computer-aided design.

Sensor data accuracy in this prototype system was limited – especially localisation data from the GPS unit. This was a critical bottleneck, and limited the type of animated patterns that could be produced. Future systems may be able to exploit rapidly evolving localisation algorithms based on Wifi, light or audio, for much higher accuracy. Opportunities for sensor fusion, to develop a clearer picture of the way the many sensors describe a contiguous environment, are yet to be exploited.

The space between pixels in distributed display systems, identified above as a characteristic of distributed system, is difficult to avoid with current pixel configurations. If the distributed pixels are arranged in a contiguous grid without any gaps it seems likely that their motility and scatterability are no longer of value, and therefore a more typical fixed-pixel based display would be more appropriate, negating the purpose of distributed displays. Various approaches to new device hardware configurations may help to solve this issue. As devices become more advanced solutions to the problem of filling these physical gaps may be developed that are optical, electro-mechanical or even as-yet unknown. However, these technologies will likely require significant further development. For the development of media transmission capabilities that are flexible and adaptable though, such a significant drawback for distributed display systems cannot be ignored, and it is likely this challenge will soon need to be solved.

Sound was an important part of the installation and was tightly coupled to the visual content. Despite the rudimentary sound synthesis capabilities of the device itself, the design palette was quite extensive, given the way that the sound complimented the visual content. Designing sound for a distributed system required a different approach than is commonly taken for sound design in public spaces. The experience of the 1000 devices making sound as opposed to 1, was quite distinct from the experience of 1000 devices making light as opposed to 1. While the visual stimulus from each device tends to be localised quite specifically within space, the nature of the auditory sense meant that the sensation of auditory envelopment was a strong characteristic of the way that the sound was experienced. This meant that the temporal complexity and variety of the sound design of the work received greater importance. Generally speaking, this leant the design process towards a probabilistic approach, where an instruction sent to the group of devices could be expected to be executed in a small but subtly different set of ways on each device, which were then experienced as a cohesive spatial experience. This proved to be quite effective, but again was difficult to simulate accurately without the installed devices.

Fusing the Computation-on-Server and Computation-on-Device methodologies to obtain optimum control over the development process may require innovative thinking and new programming paradigms. One approach is to serialise program functions and distribute them to the devices over the network, rather than sending network messages with commands and numerical parameters, in much the same way that Bown et al. have done [6]. In effect this means a distributed pixel would simply run a network listener that listens for new functions and then seamlessly executes them, requiring minimal change of the underlying firmware on each device, and no rebooting. This would also mean that the the central controller could distribute very small functions or more complex functions, and do so on a global or individual basis. As an underlying architecture, this would free the designer/developer to add or subtract interaction, animation or media content as deemed necessary (as long as they can be described programmatically), and to even do so in a real-time fashion.

However, the Computation-on-Server and Computation-on-Device approaches are not purely questions of technical architecture. They represent differing approaches to media design which will be based as much on the nature of the distributed devices as they are on the media content to be transmitted. Distributed devices in a system of this nature need not necessarily be precisely analogous to the pixels contained in a rectangular display, and they may possess other types of communication, interaction and motility capabilities that allow them to imitate and represent media content using non-typical methods. For instance, the *Spaxels* system [12] has used light-painting (using long exposure photographs) to generate 3-dimensional imagery from a small number of individual pixels. Therefore, we expect that in the future we will see not just elements of both these approaches, but also new and innovative device designs coalesce to form new media experiences within this rapidly evolving field.

CONCLUSION

Networked systems of distributed pixels represent an emerging and important new field of creativity and media design practice in digital media design. This paper has presented an account of the development of a large system of distributed pixels, and has delineated some important new capabilities in this new system. It has outlined some challenges in the process of designing media for distributed pixel systems, and has described two alternative approaches to that design process, Computation-on-Server and Computation-on-Device, comparing each carefully. Finally, this article highlighted some important challenges that designers of systems of this nature will need to address.

ACKNOWLEDGEMENTS

We acknowledge the design assistance of Digistump LLC, who designed the electronics and coordinated the manufacture of the devices. Networking support was generously provided by Xirrus Networks. *Bloom* was a SquidSoup project sponsored by the Royal Botanical Gardens, Kew, and curated by Culture Creative.

REFERENCES

1. Tom Barker, M. Hank Haeusler, Frank Maguire, and Jason McDermott. 2009. Investigating political and demographic factors in crowd based interfaces. *Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group on Design: Open 24/7 - OZCHI '09* (2009), 413. DOI:http://dx.doi.org/10.1145/1738826.1738912
2. Rick Borovoy and Brian Knep. 2011. Junkyard Jumbotron. (2011). http://c4fcm.github.io/Junkyard-Jumbotron/
3. Oliver Bown and Sam Ferguson. 2016. A Musical Game of Bowls Using the DIADs. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. 371–372. http://www.nime.org/proceedings/2016/nime2016
4. Oliver Bown and Sam Ferguson. 2017. Media Technology + the Internet of Things = Media Multiplicities. *Leonardo Transactions* submitted (2017).
5. Oliver Bown, Lian Loke, Sam Ferguson, and Dagmar Reinhardt. 2015. Distributed Interactive Audio Devices: Creative strategies and audience responses to novel musical interaction scenarios. In *Proceedings of the 21st International Symposium on Electronic Art*. http://isea2015.org/proceeding/submissions/ISEA2015
6. Oliver Bown, Miriama Young, and Samuel Johnson. 2013. A Java-Based Remote Live Coding System for Controlling Multiple Raspberry Pi Units. In *Proceedings of the International Computer Music Conference*. Michigan Publishing, University of Michigan Library, Ann Arbor, MI.
7. William Butera. 2007. Text display and graphics control on a paintable computer. In *First International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2007*. 45–54. DOI: http://dx.doi.org/10.1109/SASO.2007.60

8. Karthik Dantu, Bryan Kate, Jason Waterman, Peter Bailis, and Matt Welsh. 2011. Programming micro-aerial vehicle swarms with karma. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems - SenSys '11*. ACM Press, New York, New York, USA, 121. DOI:http://dx.doi.org/10.1145/2070942.2070956

9. Tejaswi Digumarti, Javier Alonso-Mora, Roland Siegwart, and Paul Beardsley. 2016. Pixelbots 2014. *Leonardo* 49, 4 (2016), 366–367.

10. Patrick Tobias Fischer, Franziska Gerlach, Jenny Gonzalez Acuna, Daniel Pollack, Ingo Schäfer, Josephine Trautmann, and Eva Hornecker. 2014. Movable, Kick-/Flickable Light Fragments Eliciting Ad-hoc Interaction in Public Space. *Proceedings of The International Symposium on Pervasive Displays - PerDis '14* (2014), 50–55. DOI: http://dx.doi.org/10.1145/2611009.2611027

11. Hank Hauesler, Tom Barker, and Kirsty Beilharz. 2010. Interactive Polymedia Pixel and Protocol for Collaborative Creative Content Generation on Urban Digital Media Displays. In *Proceedings of the International Conference on New Media and Interactivity*. Marmara University, 1–7. https://opus.lib.uts.edu.au/handle/10453/16609

12. Horst Hörtner, Matthew Gardiner, Roland Haring, Christopher Lindinger, and Florian Berger. 2012. Spaxels, Pixels in Space. In *Proceedings of the International Conference on Signal Processing and Multimedia Applications and Wireless Information Networks and Systems*. 19–24. DOI: http://dx.doi.org/10.5220/0004126400190024

13. Tiffany C. Inglis and Craig S. Kaplan. 2012. Pixelating vector line art. *ACM SIGGRAPH 2012 Posters on - SIGGRAPH '12* (2012), 1. DOI: http://dx.doi.org/10.1145/2342896.2343021

14. Hiroshi Ishii. 2008. Tangible bits: beyond pixels. In *Proceedings of the 2nd international conference on Tangible and Embedded Interaction (TEI '08)*. xv–xxv. DOI:http://dx.doi.org/10.1145/1347390.1347392

15. H. Ishii and B. Ullmer. 1997. Tangible bits: towards seamless interfaces between people, bits, and atoms. In *Proceedings of the 8th International conference on Intelligent User Interfaces*. 3–3. DOI: http://dx.doi.org/10.1145/604045.604048

16. Stacey Kuznetsov, Eric Paulos, and Mark D. Gross. 2010. WallBots: Interactive Wall-Crawling Robots In the Hands of Public Artists and Political Activists. In *Proceedings of the 8th ACM Conference on Designing Interactive Systems - DIS '10*. 208. DOI: http://dx.doi.org/10.1145/1858171.1858208

17. David Merrill, Jeevan Kalanithi, and Pattie Maes. 2007. Siftables. In *Proceedings of the 1st international conference on Tangible and embedded interaction - TEI '07*. ACM Press, 75. DOI: http://dx.doi.org/10.1145/1226969.1226984

18. Joseph A. Paradiso, Josh Lifton, and Michael Broxton. 2004. Sensate media - Multimodal electronic skins as dense sensor networks. *BT Technology Journal* 22, 4 (2004), 32–44. DOI: http://dx.doi.org/10.1023/B:BTTJ.0000047581.37994.c2

19. Erica Robles and Mikael Wiberg. 2010. Texturing the "material turn" in interaction design. *Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction - TEI '10* April (2010), 137. DOI:http://dx.doi.org/10.1145/1709886.1709911

20. Anthony Rowe. 2012. Within an Ocean of Light: Creating Volumetric Lightscapes. *Leonardo* 45, 4 (2012), 358–365. DOI:http://dx.doi.org/10.1162/LEON_a_00410

21. Anthony Rowe, Gaz Bushell, Liam Birtles, Chris Bennewith, and Ollie Bown. 2016. Submergence 2013. *Leonardo* 49, 4 (2016), 356–357.

22. Munehiko Sato. 2008. Particle Display System: A Real World Display with Physically Distributable Pixels. In *Proceeding of the 26th Annual CHI Conference on Human Factors in Computing Systems - CHI '08*. 3771–3776. DOI: http://dx.doi.org/10.1145/1358628.1358928

23. Munehiko Sato, Atsushi Hiyama, Tomohiro Tanikawa, and Michitaka Hirose. 2009. Particle Display System - Virtually Perceivable Pixels with Randomly Distributed Physical Pixels. *Journal of Information Processing* 17 (2009), 280–291. DOI: http://dx.doi.org/10.2197/ipsjjip.17.280

24. Arne Schmitz and Ming Li. 2010. Ad-Hoc Multi-Displays for Mobile Interactive Applications. *Eurographics* 29, 2 (2010), 45–52. DOI: http://dx.doi.org/10.1145/502348.502351

25. Julia Schwarz, David Klionsky, and Chris Harrison. 2012. Phone as a pixel: enabling ad-hoc, large-scale displays using mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 3–6. DOI:http://dx.doi.org/10.1145/2207676.2208378

26. Susanne Seitinger, Daniel S Perry, and William J Mitchell. 2009. Urban Pixels: Painting the City with Light. In *Proceeding of the 27th Annual CHI Conference on Human Factors in Computing Systems - CHI '09*. 839–848. DOI: http://dx.doi.org/10.1145/1518701.1518829

27. Anna Vallgårda, Laurens Boer, Vasiliki Tsaknaki, and Dag Svanaes. 2016. Material Programming. In *Proceedings of the 2016 ACM Conference Companion Publication on Designing Interactive Systems - DIS '16 Companion*. ACM Press, New York, New York, USA, 149–152. DOI: http://dx.doi.org/10.1145/2908805.2909411

28. Mikael Wiberg. 2014. Methodology for materiality: interaction design research through a material lens. *Personal and Ubiquitous Computing* 18, 3 (mar 2014), 625–636. DOI: http://dx.doi.org/10.1007/s00779-013-0686-7