C4: A Creative-Coding API for Media, Interaction and Animation

Travis Kirton, Sebastien Boring, Dominikus Baur, Lindsay MacDonald, Sheelagh Carpendale
Innovations in Visualization Laboratory, University of Calgary
2500 University Drive NW, Calgary, Alberta, Canada
travis@c4ios.com, sebastian.boring@ucalgary.ca,
dominikus.baur@gmail.com, macdonla@ucalgary.ca, sheelagh@ucalgary.ca

ABSTRACT

Although there has been widespread proliferation of creative-coding programming languages, the design of many toolkits and application programming interfaces (APIs) for expression and interactivity do not take full advantages of the unique space of mobile multitouch devices. In designing a new API for this space we first consider five major problem spaces and present an architecture that attempts to address these to move beyond the low-level manipulation of graphics giving first-class status to media objects.

We present the architecture and design of a new API, called C4, that takes advantage of Objective-C, a powerful yet more complicated lower-level language, while remaining simple and easy to use. We have also designed this API in such a way that the software applications that can be produced are efficient and light on system resources, culminating in a prototyping language suited for the rapid development of expressive mobile applications. The API clearly presents designs for a set of objects that are tightly integrated with multitouch capabilities of hardware devices. C4 allows the programmer to work with media as first-class objects; it also provides techniques for easily integrating touch and gestural interaction, as well as rich animations, into expressive interfaces.

To illustrate C4 we present simple concrete examples of the API, a comparison of alternative implementation options, performance benchmarks, and two interactive artworks developed by independent artists. We also discuss observations of C4 as it was used during workshops and an extended 4-week residency.

Author Keywords

Creative coding, Application Programming Interface, Mobile, Multitouch, Media, First-class objects.

ACM Classification Keywords

D.2.2 [Design Tools and Techniques]: Software Libraries. D. 2.10 [Software]: Design. H.5.1 [Multimedia Information Systems] Animations. H.5.2. [User Interfaces]: Prototyping, Graphical User Interfaces.

INTRODUCTION

The term 'media' has often been defined in a rather vague fashion: media could mean anything from images, to audio, video and 2D / 3D graphics. As developers were motivated to enable the most up-to-date types of media on devices in an efficient way, the actual implementation often relied on various background tweaks and tricks - something invisible in the application but evident to the programmer. Combining all these varied types of output under the same umbrella term therefore meant that 'media' only existed and still exists as an idealistic concept instead of as implementation term. Actually producing a media object often requires a completely different low-level approach depending on the type of object. This becomes especially apparent in programming environments and languages aimed towards artists: while striving for simplicity and accessibility, even creative-coding approaches have to force their programmers to resort to a set of different, oftentimes obscure ways of handling images, videos and 3D graphics for display, playback, interaction and animation.

With current hardware, the former distinction between various types of media has become obsolete. Faster and more powerful technology has made it possible to create an API that combines and presents all media types to the programmer in a consistent fashion, which could be especially beneficial for creative-coding languages.

To realize a media-focused and enabling environment we have developed an architecture for an API that shifts from the mechanics of working with media towards a higher-level, declarative style. We introduce C4, an API that is currently targeted for expressive interface design for mobile applications. This API is situated among other creative coding languages, however its strength is an innovative architecture that takes advantage of hardware and software systems specific for mobile devices.

After a discussion of motivation and related work, this paper is divided into four main sections: 1) the primary challenges for creating a creativity-support programming language that works with media in a higher-level 2) the design an API that addresses those challenges 3) the architecture of C4, including examples and discussion of alternative implementations 4) performance benchmarks, example artistic works, and a qualitative evaluation of the API in use through workshops and residencies.

MOTIVATION AND RELATED WORK

C4 was created to provide a stable and efficient platform for creation of expressive computational works whose interfaces focus heavily on media, interactivity and animation. As such, C4 draws its inspiration from: successful creative-coding languages, previous work on toolkit design, programming language constructs, as well as contemporary media arts.

Creative-coding languages such as Processing[22] and OpenFrameworks (OF)[21] offer great facility for learning graphic programming and flexibility for producing rapid sketches; however this ease plus flexibility approach remains to be fully extended into emerging media and technology capabilities. VVVV [27] and Max/MSP jitter [19] share this space, providing a flexible system that includes expressive opportunities for video and sound. However, all operate in the common graphics mode where full canvas refresh is used for feature animation. Replacing this with lazy-graphics update enables this type of facility for our now common high-resolution handheld devices.

The simple yet powerful design of Processing and OF were an inspiration for the growth of C4. They are both great examples of how to make programming graphics simple, however, they stop short of bringing this same power to media programming. Projects such as VVVV and Max/MSP/Jitter share influence by providing an immensely flexible system for the construction of an extremely wide variety of expressive computational works. However, it is not easy to develop native applications for mobile devices using the aforementioned projects.

Previous research has shown successful techniques for the development of toolkits and APIs. From a design standpoint the approach that GroupLab researchers have taken outlines methodologies for toolkit design, as expressed through the HapticTouch and Proximity Toolkits [17,18], Phidgets [11], as well as publications on enhancing creativity through the use of toolkits [12,13]. We apply their approach to toolkit design towards a creative-coding API. The Jazz toolkit [4] provides an interesting approach to the design of a programming language that encourages compositional approaches to programming. While innovative, the authors of Jazz acknowledge drawbacks to the system. C4 draws on their compositional technique by employing a more declarative system of control over objects. Projects such as D3, Prefuse and Protovis [6,15,24] as well as the InfoVis Toolkit [9] have provided solid examples for API design, yet are limited to the domain of interactive visualizations. We draw on their approach surrounding the discourse of C4 and by adopting successful models for describing an extensive API. Other influences in this area stem from work in API design for interactive graphics[5], and language constructs for evolving APIs [10]. Furthermore, those who have adopted mobile devices as mediums for creative expression also inspire our work. Techniques such as light painting [3] use mobile devices as brushes. The use of mobile devices as canvases as well as a means for exhibition is becoming widely accepted in the art world[25]. These highlight the ability to move into an application development space that encompasses the use of devices as objects for creative exploration and expression.

PRIMARY CHALLENGES

We identified six primary challenges for the development of a creative-coding language that works with media in a higher-level, declarative style, as follows:

Media Objects. Media – videos, images, audio, text, and shapes – have not yet received first class status in programming but largely remains accessible via device level programming. That is, developers manipulate containers of media (e.g., a *view* hosting a video) instead of the media itself. A negative side effect is that each of these containers is treated differently depending on the type of contained media. For example, a developer has to access a movie in a different way than s/he would access a shape or text. This has probably arisen out of the step-by-step integration of new media types in existing APIs (i.e., video was added later than images).

Media Integration. As with media objects, the integration of media with other programmable objects such as graphics and interface components remains similarly low-level. This challenge increases with the desire to have disparate media affect and influence one another. In many cases, programmers may need to know low-level techniques. For instance, blending two images might necessitate the use of a graphics processing language such as OpenGL [21].

Interaction. In current languages, interaction with media objects is enabled through their enclosing containers. That is, a media object does not handle the interaction directly. With the addition of multi-touch and gestural interaction, the problem is further amplified as they define a new interaction style. Most languages and APIs, however, were written for systems that relied primarily on mouse interaction requiring the developer to build touch and gesture recognition systems from scratch.

Animation. Although animating media objects is possible in existing APIs, they have to be built differently depending on the type of media to be animated. The problem is that developers spend a lot of time dealing with the mechanics of the application as opposed to the putting more focus on the application's state. Furthermore, in most APIs, there is no unified mechanism for animating different objects. For example, animating a shape is different from animating a sequence of images and so on.

Rapid Mobile Development. While mobile development is proliferating, creating and setting up a project is not as straightforward as on desktop computers: developers have to understand several (mostly device-specific) abstract concepts, such as application initialization hierarchies. Furthermore, mobile application development commonly necessitates coding in the language specific to the device. This implies needing to know many native implementations and media-specific technologies, something that can be daunting for novice and intermediate level programmers.

Efficient Rendering. Another significant issue that arises for applications intended for mobile use is that they have to run as efficiently as possible. Current APIs, however, update the entire screen even if there is nothing that needs to be updated in a given cycle. With mobile devices this is a serious problem given their limited power resources.

DESIGNING A CREATIVE-CODING API FOR MEDIA

To design an API that addresses the aforementioned challenges, we took a two-step approach: (1) *Design requirements* that seek to shape the experience of working with the API from the programmer's perspective; (2) *Software requirements* shape the architecture of classes and class structures in such a way that enables and reifies the design ideals and provide basic techniques for guiding the creation of the code that is embedded into the API. These steps serve to direct the overall development of the project from conceptual, architectural and engineering perspectives.

Design Requirements

Other APIs have been successful in implementing easy to use languages for programming graphics. However, our goal was to do so for media objects, interaction and animation. Based on the identified challenges, we created a list of *ideals* that better fits the expectations of a developer working with the API. Most importantly, we set out to consider things that a developer would want or expect from other mature APIs, such as ease and consistency of use. In particular, the *ideals* are:

Media as First-Class Objects. Video, voice, music, and images, should have first class programmable status. That is, programmers should be able to work directly with media instead of their encapsulating containers. All media properties should be accessible programmatically, allowing developers to control and manipulate any state of any media object directly. The access through a media object's properties and the corresponding response should be consistent across various media types.

Easy Integration and Composition. Media should be easily integrated with another one – regardless of the involved media types. This allows for adding shapes to movies, text to images, and so on, creating the possibility for designing complex media types. This integration should also allow for media composition, e.g., masking a video with a shape.

Direct Interaction with Media Objects. Developers should be able to construct interactive interfaces – including multitouch and gesture – from any type of visual object. An API should take full advantage of underlying frameworks, removing the need for developers to build touch and gesture recognition systems.

Declarative Animations. A developer should be able to animate each media object directly by setting target states and transition styles instead of constructing the mechanics of an animation. This declarative style should decrease the time spend on constructing complex animations.

Rapid Development. Instead of focusing on the setup of a project, developers should be able to rapidly prototype media rich applications. Thus, the time to get a project up and running should be kept short, i.e., through the use of easily installable and useable templates.

Efficiency. Media objects should operate as efficient as possible. Instead of updating a screen's content on a frame-by-frame basis, the API should make use of lazy rendering.

Software Requirements

The overall goal of our architecture is to directly support the ideals of the project. Also, the same principles for working with media should be reflected in the way code is used and designed. The following list of goals identifies targets for the concrete aspects of the API and its implementation. These targets help determine whether new additions, add-ons or fixes conform to the expectations of developers of the API.

Simplification. The architecture should be as simple as possible. To minimize the conceptual design of the API, we simplify media objects into two main classes: visual and non-visual objects (Figure 1).

Use of Properties. Developers should be able to control media objects and their states through properties. When properties for a given object do not exist in the native language, the API should implement pseudo properties.

Unified Methods. Whenever possible, the API should wrap common operations into single methods. In current APIs, several operations exist that are only slightly different from one another. Our API provides single operations instead of many slightly different operations.

Direct Use. In our API, media objects are designed as composite structures. The intention is encapsulate native objects and work with them the way they were designed, rather than sub-classing and extending their functionality. We also use existing frameworks to make use of existing, complex technologies as opposed to reinventing them.

Familiarity. We intend to make the API look like the underlying language. That is, the API reflects existing name conventions as well as programming styles of Objective-C, the language with which C4 was built. This will help developers use native code in their applications, should they need more custom functionality.

THE ARCHITECTURE OF C4

The C4 framework is written in Objective-C / Core Foundation and is deployed on iOS environments (i.e. iPad and iPhone) as these devices are commonly used for application development. As such, the development environment is Xcode and the application development process is similar to that of creating native iOS applications. Furthermore, while it may be possible to develop C4 for other platforms (e.g., Windows Phone, Android) the current version demonstrates how our streamlined API realizes the aforementioned ideals.

The architecture of C4 conforms to our stated requirements and, in doing so, seeks to address the six challenges for creating a creativity-support programming language. From a design perspective, C4 offers two class-clusters from which all objects descend; these clusters, C4Object and C4Control, provide interfaces for high-level interaction with visual and non-visual objects. Two strong examples of integration are that all objects can communicate with one another via a simplified notification system, and that any visual object can be used as a mask for any other. The design of visual objects employs an architecture which addresses both direct interaction and efficiency by inheriting the touch / gesture interaction capabilities of native views, as well as as-needed rendering. This means that a visual object's contents are updated / redrawn only when absolutely necessary. Finally, all objects incorporate the use of properties to control state and where these properties are animatable; simply setting a new target value will trigger an animation.

The overall goal of C4 is *simplification*. A programmer will generally be dealing with two types of objects, visual and non-visual, that share common functionalities (Figure 1). Where the native API presents similar methods for a common operation, C4 attempts to encapsulate such functionality into unified methods. For example, the native use of gesture recognizers requires knowing seven different objects that share common methods for customization, C4 simplifies this into two methods used to construct and customize all gestures. Class design further adopts a composite structure directly with the use of native objects and frameworks. Two primary examples are C4Movie, which provides an interface for working with a native video player, and C4Vector, which wraps calls to the Accelerate framework [1]. Finally, C4 is structured to look like

Objective-C through method naming conventions and other techniques. This provides a familiarity that allows novices to gain an understanding of the underlying language, making any` eventual transition into native application development easier.

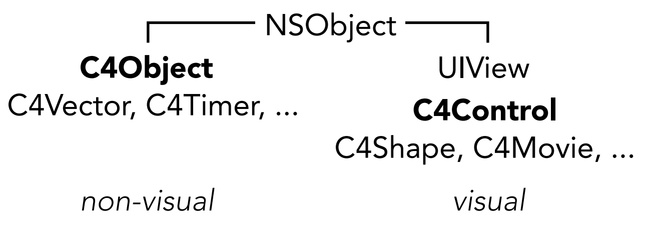

Figure 1. The class structure of objects is simplified into visual and non-visual objects

EXAMPLES

In order to illustrate some of the basic capabilities of C4 we present six short examples that reflect the listed problem spaces. We recognize that while it may be difficult to grasp the full scope of the API from these examples, they do express the simplicity of the language. Furthermore, the simplicity of these examples do not limit the creation of more complex setups. C4 can be used at a higher level by experienced developers; it can also be integrated as a pre-compiled library into native iOS projects – doing so makes it easier to construct basic media objects instead of writing complex native code.

Adding to the Canvas

The following creates a shape and adding it to the canvas:

```
CGRect r = CGRectMake(0,0,300,300);
C4Shape *s = [C4Shape ellipse:r];
[self.canvas addShape:s];
```

A CGRect structure, r, defines a space within which the shape will be created. A shape object is constructed in the space defined by r and is then added to the canvas. For all subsequent examples, the reader can assume that *any visual* object is added to the canvas in a similar fashion.

Animating a property: lineWidth

Properties provide simple means for customizing and controlling objects. Many properties of visual objects are animatable; the following example illustrates animating the line width of a circle (Figure 2).

This animation is triggered using the following code:

```
s.animationDuration = 1.0f;
s.lineWidth = 150.0f;
```

The `animationDuration` is a property of all visual objects that determines the length of any transformation. This example is creates a one-second animation. Any animation is preceded by setting the object's animation duration.

Morphing Shapes

All shapes can implicitly transform, even between normal polygons and text shapes. The following example illustrates changing a circle to a square (Figure 3).

This animation is triggered using the following code:

```
[s rect:s.frame];
```

The animation system does its best to interpolate between the two shapes.

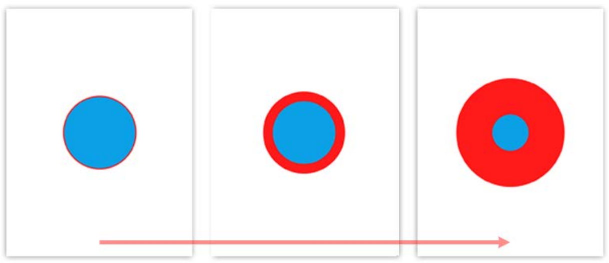

Figure 2. Animating (left to right) the line width of a shape.

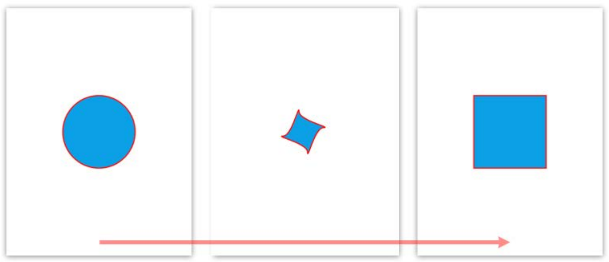

Figure 3. Morphing a shape from an circle to a square.

Creating and Animating Images

Creating an image is quite easy; the programmer need only specify the name of the image file.

```
C4Image *i = [C4Image imageNamed:@"C4Sky.png"];
```

Animating an image is nearly the same as animating a shape. In fact, all properties for all visual objects behave in similar fashion.

All images have a width property that can be animated. An image can be fitted to the size of the screen by referencing the width property of the canvas (Figure 4).

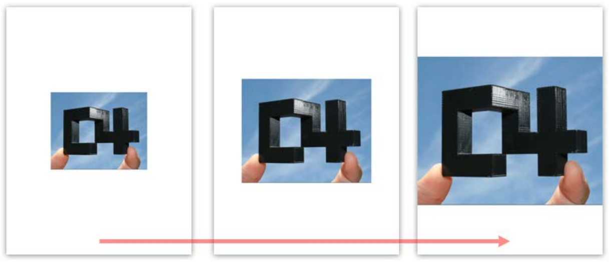

Figure 4. Animating the width of an image.

```
s.width = self.canvas.width;
s.center = self.canvas.center;
```

The second line in this example makes sure that the image stays in the center of the screen as it animates. It also shows that the canvas and the image have a similar property called *width*. The difference between the two is that the width of the canvas cannot be changed.

Image Filters

There are thirty-eight different filters that can be applied to an image. Some filters require a second image to create a blend, while others only require input values to change the look of the image. The following is a simple example of using an input-style filter to change an image (Figure 5).

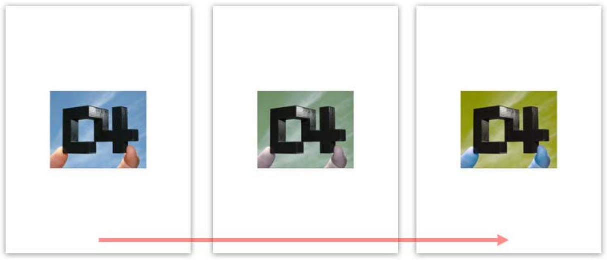

Figure 5. Applying a hue filter to an image.

```
[i hue:10.0f];
```

Calling the *hue* method on an image will change the color of the image, and if the duration of the image was properly set the transition will animate.

Labels and Fonts

Labels are visual objects that have properties and behave like images and shapes. Fonts are non-visual objects that can be applied to labels, or to shape objects making them look like text (Figure 6).

```
C4Label *l = [C4Label labelWithText:@”Hello C4”];
C4Font *f = [C4Font fontWithName:n size:120.0f];
l.font = f;
```

This example specifies *n* as the font name but in reality can be any one of the fonts available on iOS. The above actually uses `@”ArialRoundedMTBold”` in place of *n*.

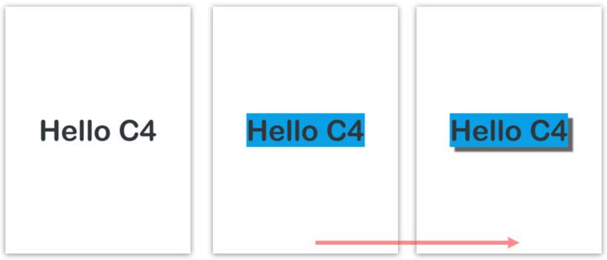

Figure 6. A label (left), background color (middle) and an animation of the label’s shadow (right).

The background of the label can be changed, as follows:

```
l.backgroundColor = C4BLUE;
```

To apply a drop shadow:

```
l.shadowOpacity = 0.8f;
l.shadowSize = CGSizeMake(10,10);
```

Movies and Gestural Interaction

Movie objects can be created in the same way as images, by specifying the name of a file. All visual objects have the capability for gesture recognition, in the case of a movie gestures can be used to control its playback (Figure 7).

Gestures can be added to an object by passing a gesture *type*, this example uses the `TAP` gesture to control playing and pausing the movie. The first gesture, `gest1`, will trigger the movie’s play method; by default the TAP gesture requires only a single touch. The second will trigger the pause method, but only if two fingers tap at the same time.

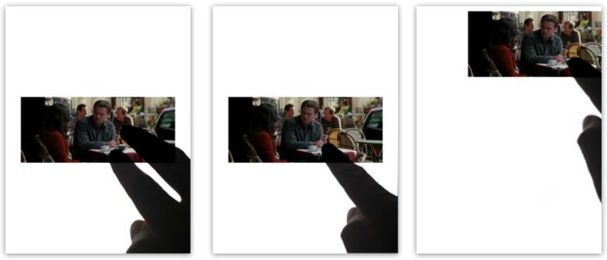

Figure 7. Two-finger tap (left) will pause the movie, single-finger tap (middle) will play the movie, and a single-finger drag will displace the movie.

```
C4Movie *m = [C4Movie movieNamed:@”file.mov”];
[m addGesture:TAP name:@"gest1" action:@"play"];
[m addGesture:TAP name:@"gest2" action:@"pause"];
[m numberOfTouchesRequired:2 forGesture:@"gest2"];
```

An object can be dragged around the screen by adding a PAN gesture and having it trigger the move: method. The following line does so for the movie in this example:

```
[m addGesture:PAN name:@”gest3” action:@”move:”];
```

Alternative Implementations

The best example of this is the `C4Movie` object. Forgoing the code to set up the application environment, as well as defines, imports and counting the header definitions (all of which are provided by C4) as well as individual method declarations, a comparison of lines of code to construct and add a movie to the canvas is as follows:

- C4: 2 lines of implementation
- Native: 103 lines of implementation

C4 simplifies the process for adding *any kind* of gestural interaction to objects:

- C4: 2 lines of implementation
- Native: 47 lines of implementation

Other creative-coding languages

It is also possible to write these examples in other creative-coding languages. However, many such languages are less efficient for dealing with interaction. A programmer using one of these APIs would have to build a custom event handling system and write touch / gesture recognition logic. C4 exploits underlying hardware-software frameworks making it suited to handle multitouch interaction.

In many cases, a programmer using a graphics-based API with a common *draw-loop* architecture would have to code every step of an animation. This makes the combination of various animations with differing start times quite difficult. C4 allows for the combination of many animations by calling them, or setting various properties, at the same time.

Finally, C4 employs an *as-needed* approach to drawing. This means that the system only redraws when it needs to do so and only in areas of the screen that need to be updated. In order to match this efficiency without C4, a programmer would have to further construct mechanisms to manipulate and reduce the inherent frame-rate mechanisms that make drawing calls and render the canvas.

PERFORMANCE BENCHMARKS

By employing an as-needed rendering system C4 greatly reduces unnecessary computation. Furthermore, being written in code designed for the hardware environment on which the API is running allows for each application to be compiled tightly into an architecture suited for the platform. These aspects offer improved performance over other creative-coding languages that are designed to be cross-platform, and can run at or very near the efficiency of those written in the API's native language. We now substantiate these claims by presenting performance statistics for four applications implemented in three different contexts (Figure 9).

Four applications, Three Languages

To test our API we designed 4 simple applications that each focus on a single type of media. The applications do the following: 1) draw a circle, 2) draw text, 3) draw a scaled image and 4) draw a movie (Figure 4).

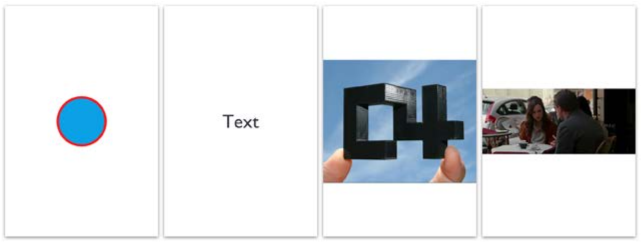

Figure 8. Four applications written in C4, ObjC and OF for testing performance

Each application was implemented in C4, ObjC and OF (totaling 12 apps). OF is a popular creative-coding API that can run on mobile devices, and specifically iOS. We compare implementations of simple OF examples that are given out-of-the-box by replicating them in C4. In doing so, we are essentially comparing our implementation to the simple cases provided by the OF project. This approach does not undermine the validity or functionality of OF, but it does position C4 against one of the most popular and successful creative-coding APIs. Furthermore, as many other APIs, such as Processing, are not available for iOS we cannot benchmark their performances. Also, C4 has not been implemented on Android and so benchmarking between hardware platforms cannot be made consistent.

Each application was tested in three different contexts – CPU activity, OpenGL frame count, and total memory allocation – totaling 36 individual metrics. Measurements for each application were started at launch and taken for 30 seconds. Testing was done on an iPhone 4S, running iOS 5.1 and recorded using the Instruments developer tool.

CPU Activity

The CPU activity for each application was measured against the overall system CPU. In all cases, the beginning of each trace showed a spike in activity as an application

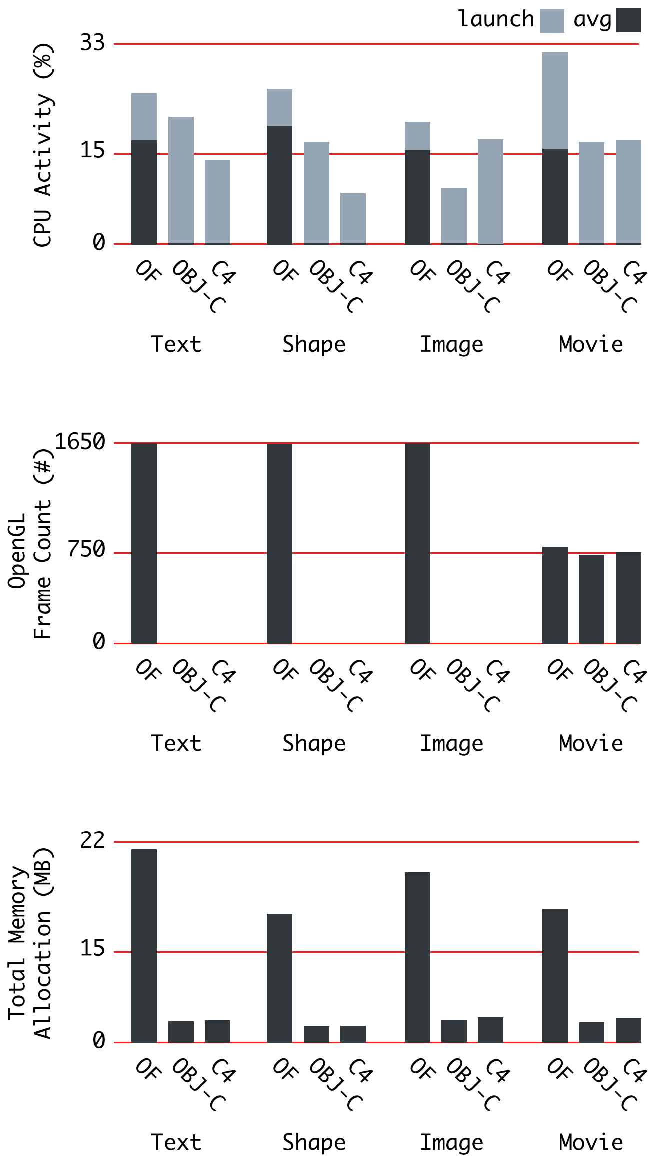

Figure 9. Performance benchmarks for C4 against ObjC and OpenFrameworks

was starting up, afterwards the activity dropped significantly. The OF implementation showed the highest overall CPU activity and consistently higher average activity after launch. This is most likely because of its draw-loop architecture. The ObjC and C4 implementations showed varying launch activities, but negligible averages thereafter, between 0.1-0.32%, across all four applications.

OpenGL Frame Count

Comparing OpenGL frames drawn shows that ObjC and C4 applications used zero calls to OpenGL, except in the case of the movie application, showing that both ObjC and C4 do not continuously render. OF rendered approximately 55fps for the first three applications. The movie application ran at ~25fps for all three implementations.

Total Memory Allocations

This benchmark illustrates the volume of total allocations of memory. For all applications the ObjC and C4 implementations measured between 1.79 and 2.64MB of memory allocated. C4 showed negligibly higher allocation than ObjC. The shape application had 0.06MB discrepancy compared to 0.36MB for the movie application.

Discussion of Performance Benchmarks
The performance data shows three important aspects: 1) C4 shows consistently low levels of activity for working with basic media 2) the activity of C4 compares with ObjC suggesting it be nearly as efficient as native code 3) C4 can handle a significant increase in OpenGL frame count without displaying marked increases in either CPU activity or memory allocations. Though these preliminary results are very promising, a full suite of tests checking various performances would be necessary to be definitive.

INTERACTIVE ARTWORKS MADE WITH C4
The following are strong examples of artists using C4 to create new interactive artworks. Each of the artists were new to using C4, and each project was created by them over the course of a 4-week residency (Figure 10). Artists were directed to express their unique creative vision.

Black Sheep, by Manuel Ermecheo, is an interactive video portrait using 6 devices to display videos of a person's face [8]. Each device displayed a specific portion of a face and contained 4 videos of the same portion from 4 different people. The videos could be changed via swipe gestures to animate between videos. *(32 Lines of code)*

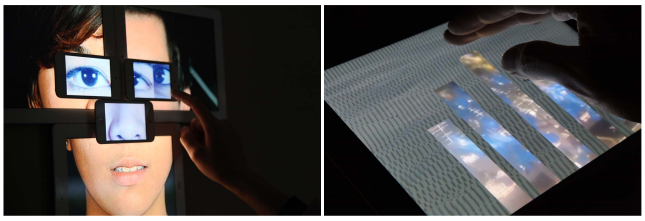
Figure 10. *Black Sheep*, by Manuel Ermecheo (Left). The RedC(4) by Lindsay Sorell (right).

The Red C(4), by Lindsay Sorell, is a dynamic interactive video-masking project using a combination of 3 overlaid videos [26]. Gestures are used to initiate control over a masked draggable video. Sound is also controlled based on interaction, fading various audio samples in and out. Animations return the work to its original state when interaction is ended. *(90 Lines of code)*

C4 IN USE: WORKSHOPS
C4 has been used extensively in two different scenarios: a 3-week session of short workshops and a 4-week residency for a small group of artists. Throughout these sessions gathered informal observations about the C4 project.

A group of approximately 45 participants with little to no programming experience from the Alberta College of Art + Design were introduced to C4 as part of a series of single-day workshops. Participants were tasked to create an animated interactive work as either a poem or a branding project for their favorite band.

DISCUSSION
The two major sessions, totaling 7 weeks focused on the use of C4 by approximately 50 different people. These helped to solidify the API and the final projects produced affirmed the goals of C4 and its effectiveness as a creative-coding API. We frame this discussion around our design requirements.

Challenges
Media as First-class Objects. The artists made extensive use of the API's treatment of media as first-class objects. The declarative control over the state of each object seemed to provide the intended freedom. It supported use of media in ways we had not expected by providing higher-level interaction allowing the artists to focus more on expression rather than on developing the mechanics of their works.

Easy Integration and Composition. The participants made complex integrations of various disparate media objects with one another. The major components of this aspect are: observation / communication, the incorporation of visible objects into one another, masking and filtering.

Interaction and Animation. Many aspects of visual objects were both used: 1) as interface components with added touch and 2) gestural recognition. These animations were used in response and independently

Rapid mobile development. All participants made applications for either touch pads or touch phones. While the participants were novice programmers, all were pleasantly surprised by their 'own programming abilities' and by the projects they produced. This is perhaps the best affirmation of C4 – the participants clearly felt empowered.

Quick. The start time to building a mobile application was amazingly short. After running the installer package, workshop participants were able to compile and test their first projects within minutes.

Learnable. All participants were able to develop interactive mobile applications on their first day of using C4. The majority of participants from the workshops were first and second-year art school students who had never taken a programming course.

Expressive. The simplicity of the C4 allowed participants to focus more on expressing their artistic intention instead of on how to actually make something happen. This allowed them to focus more on working with media rather than on the mechanics of various kinds of media.

Efficiency. This is a low-level feature that stood up to the needs of the projects but was not often commented upon by participants.

Lightweight + Robust. During the exhibition of the four works from the residency, 12 devices each ran interactive applications for more than 5 hours without being connected to power sources. The API is efficient enough to be run for long periods of time on mobile devices; a major goal.

Accessibility / generality. One of the most significant aspects of C4 is the fact that it dramatically reduces lines of code needed to produce an application.

FUTURE WORK
We see a strong possibility for expanding the control of C4 applications to tangible media, building bridges to hardware devices such as Arduino controllers. We are interested in the potential of using Media Objects for interactive Data Visualizations. Finally, we see the possibility of developing C4 for other platforms, such as the Windows Presentation Foundation. We are also looking

towards the addition of simplified application development components – including various kinds of views, windows and object controllers – that share common characteristics of existing visual objects.

CONCLUSIONS

We have presented C4, the implementation of a creative-coding API with media as first-class objects integrated with readily available animations and touch and gesture interactions. C4 is a rapid prototyping language for experimenting with mobile tablets. Throughout its development, C4 was strongly influenced by the needs of artists and designers for an expressive API. The major API design contributions of C4 are:

First-class media. The API treats media as first-class objects with declarative control over the state of each object. This provides higher-level interaction allowing the programmer to focus more on expressing intent rather than on developing the mechanics of working with media.

Media Integration. The API offers numerous ways for the programmer to integrate various disparate media objects with one another. The major components of this aspect are: observation / communication, the incorporation of visible objects into one another, masking and filtering

Composite Objects. The structure of individual classes is flexible; the focus is on encapsulation rather than subclassing. Composite objects contain and provide access to native objects. This allows for grouping objects into distinct class clusters, visible and invisible, creating a more uniform and consistent interface between objects.

Interaction. Visible objects are interface elements to which touch and gestural recognition may be added.

Animation. The API employs a strong implicit animation model that makes it easy to create and control animations through properties and declarative statements.

Rapid Mobile Development. C4 offers straightforward templates for setting up projects; it simplifies needed code, and lessens the need for knowing many media-specific technologies. It has been show as accessible for novice and intermediate level programmers.

Efficiency. Project installations illustrated the usefulness of efficiencies of C4 such as the use of lazy updating of a screen's content.

ACKNOWLEDGMENTS

We would like to acknowledge: Manuel Ermecheo and Lindsay Sorell, for the presentation of their artworks; The Alberta College of Art + Design hosting workshops; and NSERC, SMART, AITF, SurfNet, GRAND, and University of Calgary for support.

REFERENCES

1. Apple Developer. The Accelerate Framework. https://developer.apple.com/library/mac/#documentation/Accelerate/Reference/AccelerateFWRef/_index.html
2. Arduino. http://www.arduino.cc/
3. BERG. http://berglondon.com/blog/2010/09/14/magic-ipad-light-painting/
4. Bederson, B., et al. 2000. Jazz: an extensible zoomable user interface graphics toolkit in Java. In *Proc. UIST* (2000). ACM, New York, NY, USA, 171-180.
5. Bederson, B., Grosjean, J, Meyer, J. Toolkit Design for Interactive Structured Graphics. *IEEE Trans. Softw. Eng.* 30, 8 (2004), 535-546.
6. Bostock, M., Ogievetsky, V., Heer, J. D3 Data-Driven Documents. *IEEE Trans. on Visualization and Computer Graphics* 17, 12 (2011), 2301-2309.
7. Core Animation. https://developer.apple.com/technologies/mac/graphics-and-animation.html
8. Ermecheo, M. Black Sheep. https://vimeo.com/44775468
9. Fekete, J-D. The InfoVis Toolkit. In Proc. of the *IEEE Symposium on Information Visualization* (2004), 167-174.
10. Freeman, S., Pryce, N. Evolving an embedded domain-specific language in Java. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* (2006), 855-865.
11. Greenberg, I., Kumar, D., Xu, D. Creative coding and visual portfolios for CS1. In *Proc. of the 43rd ACM technical symposium on Computer Science Education* (2012), 247-252.
12. Greenberg S., Fitchett, C. Phidgets: easy development of physical interfaces through physical widgets. In *Proc. UIST* (2001), 209-218.
13. Greenberg, S. Enhancing Creativity with Groupware Toolkits. Invited keynote talk. In *Proc. of the CRIWG' 03 9th International Workshop on Groupware* (2003), LNCS vol. 2806, 1-9.
14. Greenberg, S. Toolkits and interface creativity. *Multimedia Tools Appl.* 32, 2 (2007), 139-159.
15. Heer, J., Card, S. K., Landay, J. A. prefuse: a toolkit for interactive information visualization. In *Proc. SIGCHI* (2005). 421-430.
16. Instruments. https://developer.apple.com/technologies/tools/
17. Ledo, D., Nacenta, M. A., Marquardt, N., Boring, S., Greenberg, S. The HapticTouch toolkit: enabling exploration of haptic interactions. In *Proc. TEI* (2012), 115-122.
18. Marquardt, N., Diaz-Marino, R., Boring, S., Greenberg, S. The proximity toolkit: prototyping proxemic interactions in ubiquitous computing ecologies. In *Proc. UIST* (2011), 315-326.
19. Max/MSP. http://cycling74.com/products/max/
20. OpenGL. http://www.opengl.org
21. Openframeworks. http://www.openframeworks.cc/
22. Processing. http://processing.org/
23. Pure Data. http://puredata.info/
24. Rostock, M., Heer, J. Protovis: A Graphical Toolkit for Visualization. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (2009), 1121-1128.
25. Royal Ontario Museum. David Hockney. http://www.rom.on.ca/hockney/artist.php
26. Sorell, L. The Red C(4). https://vimeo.com/44784050
27. VVVV. http://vvvv.org/