Finger Paint: Cross-platform Augmented Reality

Samuel Grant Dawson Williams

October 15, 2010

Abstract

Finger Paint is a cross-platform augmented reality application built using Dream+ARToolKit. A set of high level augmented reality interfaces have been developed which allows the same application code base to be compiled for use on both desktop and mobile computers. The main goal of this work is to reduce the cost of augmented reality application development and testing for multiple devices and platforms.

Contents

1 Introduction

Mobile computers – including phones and tablets – present a unique opportunity for augmented reality applications due to their convenient size, configuration and availability[1]. A typical modern mobile computer includes one or two cameras, a touch screen, and a reasonable amount of processing power. However, despite significant advances, the hardware available in mobile computers can still requires specific algorithms and software engineering knowledge to ensure a reliable and responsive applications. This is especially true for intensive graphical rendering and image processing where large amounts of data are processed in real time.

Augmented reality applications typically involve image based tracking and processing algorithms as well as complex interactive geometry overlaying video input. Traditional augmented reality desktop applications might not perform well when ported directly to a mobile computer, because of the reliance on desktop class hardware and assumptions about connectivity. Also, many mobile computer platforms expose proprietary software stacks which are not compatible with each other.

Because of these problems, it can be costly to create a cross-platform mobile computer applications that work reliably, especially one that relies on advanced algorithms required by augmented reality.

1.1 Definitions

Computer
: A computer includes any kind of device designed for executing software programs. Computers generally provide some kind of storage and processing capabilities, including specifically user input, graphics and audio.

Desktop Computer
: Desktop Computers typically refer to a large powerful computer with two or more processing units.

Mobile Computer
: Mobile computers typically refer to low power devices with reduced processing capability and include devices like cell phones and tablet computers.

Cross-Platform
: Cross-platform is a quality which means that something can work on a variety of different hardware and software platforms with minimal changes or effort.

iOS
: The name of the operating system developed by Apple Inc for the iPhone, iPod and iPad collective of devices. It is based on existing technologies from Mac OS X and BSD.

1.2 Motivation

Developing applications for mobile computers is not always easy. There are many different hardware configurations, software platforms and input mechanisms. The typical code-compile-test cycle is far more complex:

- Different software platforms for different devices require developers to learn new interfaces and techniques. Specific devices may even require completely different programming languages ensuring that effort for one platform is not transferrable to another.

- Cross-compiling software for different platforms and devices is complex due to the inherent reliance on platform specific data structures and interfaces. Low level dependencies often have different build procedures and this can make it almost impossible to integrate libraries in a way that reduces the overall complexity of a software project.
- Testing and debugging is often far more complex due to limited connectivity and feedback. Compounding with all the previous points, testing relies on good debugging tools, and furthermore, these must be able to work remotely in order to provide sufficient usability when debugging large applications.

Modern software development paradigms do not work well in conjunction with mobile computers because of the high level of coupling with device specific features and the complexity of integrating a wide range of technologies for a specific device.

However, as mobile computers become more powerful, it is becoming easier to write general code that works on multiple devices including desktop class hardware. This includes general interfaces for developing cross-device applications and libraries.

Dream[2] is a project that I have been working on for several years. It combines many different APIs into a single library designed for game development. It compiles on multiple platforms easily and reduces the amount of work required to develop software for multiple devices.

1.3 Goals

The Dream framework aims to provide a cohesive platform for event-driven game development:

- Cross-device build and development support from a single code base.
- Reference counted pointers for memory management.
- Event driven networking and rendering engine.
- A robust message based communication framework.
- Vector, Matrix and Quaternion mathematics.
- Sphere, Line, Plane, Frustum, AlignedBox and Triangle for geometry calculations.
- Integration with FreeType2 for text rendering.
- Positional audio using using OpenAL and Ogg Vorbis.
- OpenGL for cross-platform graphics (including support for OpenGL and OpenGL ES).
- LibPNG and LibJPEG for image loading/saving.

This particular project, Finger Paint, is a single application using the Dream framework and integrates with ARToolKit to provide marker tracking. It has several goals:

- Extend the Dream framework to support the video capture as provided by ARToolKit.
- Experiment with the required mathematical constructs to track markers.
- Expose a simple finger based input for drawing lines on the plane defined by a marker.

It is expected that experience in this field will extend to many different cross-platform challenges.

2 Background

2.1 Existing Work

There are many existing software stacks which include support for application development on mobile computers.

Wagner and Schmalstieg have developed a large number of mobile applications including a full software stack called Studierstube[3]. They draw on their considerable experience with mobile computers to discuss both approaches to software architecture[4] and the impact hardware limitations have on software development[5]. Furthermore, they suggest that software specifically developed for mobile computers can deliver far better performance than software ported from traditional desktop computers.

The Oolong Engine[6] integrates a wide variety of software libraries into a single project specifically designed for iOS development. It is released under a open source software license and has been used by many existing projects to make interactive games. Due to its open ended design, it does not have any consistency over the entire framework.

SIO2[7] is a commercial game engine which provides some cross-device development features. It integrates with a wide range of libraries and as part of the commercial agreement includes many examples and game assets. As a developer, SIO2 also provides a market place for source code.

Impact[8] is a JavaScript based game engine that utilises the cross-platform features of JavaScript and the HTML5 <canvas> element. In addition to this, it can also function as a stand alone game engine and this has been shown to work efficiently on iOS. This software stack provides a consistent cross-platform environment for developing graphically advanced applications, and is supported by the availability of high performance JavaScript interpreters for many different platforms.

2.2 Relevance

A primary influence on the development of mobile software technology is the power of the underlying hardware; as mobile computers continue to increase in power, many previous approaches are no longer necessary or useful. Application software which targets specific device and platform configurations can become obsolete as new devices are released, and this may increase the costs associated with software development.

Of the above approaches, Impact is the only feasible long term approach to software development. This is because software written in a portable high level language is an asset. Conversely, software written in a device dependant language might be considered a liability.

3 Implementation

I am working on integrating an existing augmented reality tracking system called ARToolKit into a game engine for use with mobile computers. ARToolKit is available on several different platforms, including iOS and provides a medium-level set of interfaces for marker tracking.

I have also been working on a high-level game engine called Dream since 2006. Part of this process involves understanding the nature of cross-platform development and ensuring that the interfaces provided are sufficiently useful and flexible, while minimising the performance impact, if any.

Dream (see figure 1 for an overview) has been developed from the ground up to be portable, but only with desktop computers in mind. In 2009, I began porting Dream to the iOS (iPhone) platform as an experiment; this process is streamlined because of my prior experience with Mac OS X (as a development platform). However, my experience of mobile computers' video interfaces and ARToolKit is limited, and this knowledge will be important in order to develop a cohesive high level abstraction.

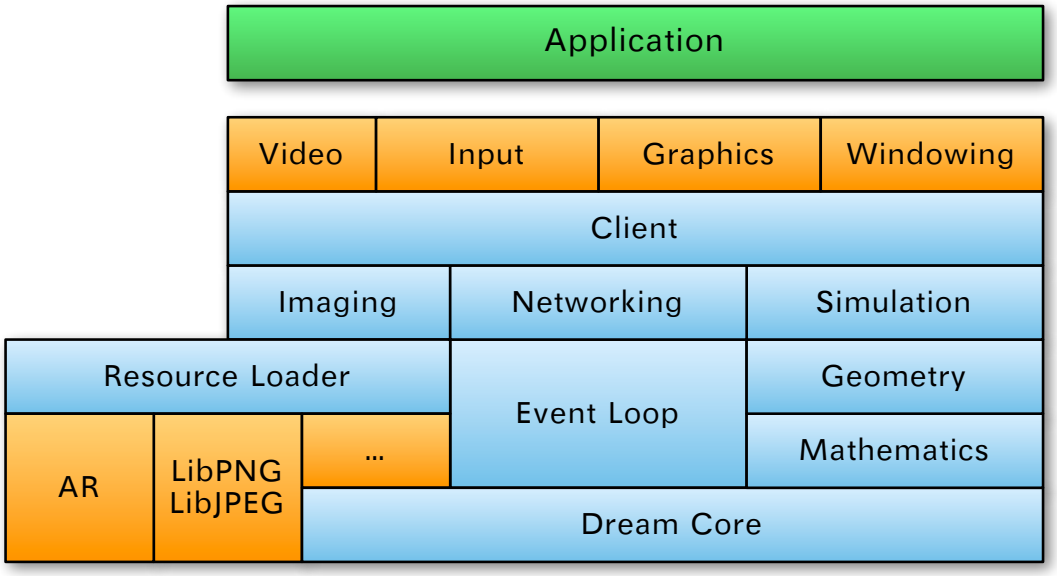

Figure 1: A block diagram of Dream. Green areas show application code. Blue areas show Dream framework code. Orange areas show platform dependent abstract interfaces.

3.1 ARToolKit Integration

One of the primary challenges of this integration project was the various different configurations of ARToolKit and the way it must be used. There are two different versions of ARToolKit, version 2, and version 4. There is a specific build of version 4 for the iOS platform.

Because of this, a large number of preprocessor directives were used in order to isolate the different APIs where required. However, most of these are part of the interface implementation and not exposed to the application developer. An overview of the implementation can be seen in figure 2.

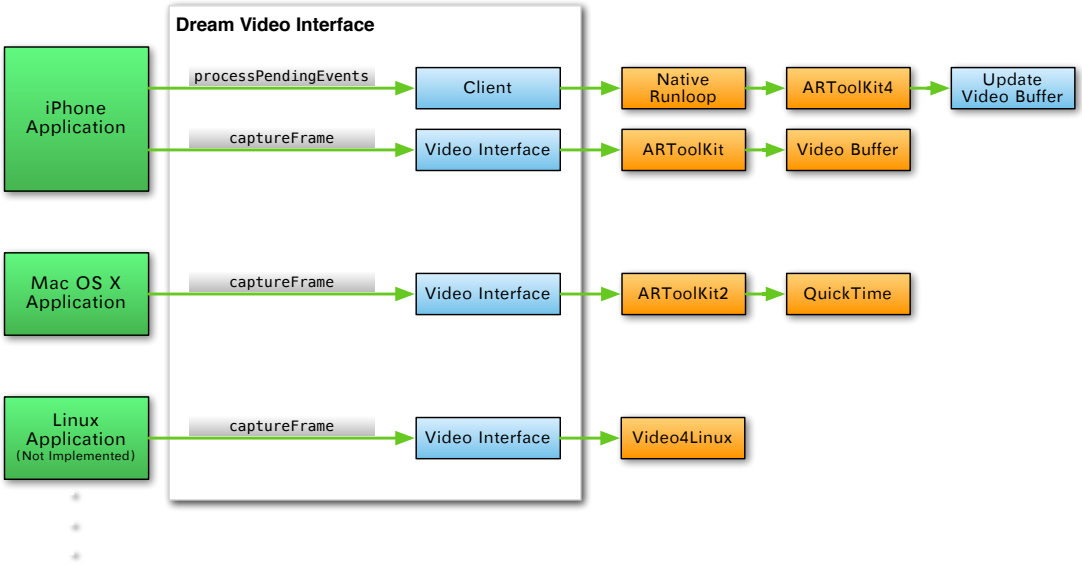

Figure 2: A high level overview of the video interface.

3.1.1 Host Integration

Because ARToolKit integrates with the host system in different ways, this had to be supported transparently with the existing high level interfaces. This is done by hooking the video in through the event loop on the iPhone platform, which doesn't require any changes to existing code.

Listing 1: Integrating with the iOS run-loop

```
void Context::processPendingEvents (IInputHandler * handler)
{
    g_inputHandler = handler;

    SInt32 result;

    do {
        result = CFRunLoopRunInMode(kCFRunLoopDefaultMode, 0, TRUE);
    } while (result == kCFRunLoopRunHandledSource);

    g_inputHandler = NULL;
}
```

3.1.2 Application Integration

The end user simply has a single function to call to get a video frame:

Listing 2: Reading a frame from the video input

```
REF(IPixelBuffer) videoFrame = m_videoInput->captureFrame();
```

Once a video frame has been captured, it can then be processed by ARToolKit to detect markers. This is done using the following interface:

Listing 3: Detecting markers in the video input

```
// Detect markers from frame.
std::vector<ARMarkerInfo> markers;
m_videoInput->detectMarkers(50, markers);
```

3.2 Run Loop

Many software stacks provide their own run-loop which provides a central area for event handling in the application. Dream provides its own high performance run-loop designed around **kqueue** or **epoll** depending on what is available.

The Dream run-loop integrates with other systems through the user input mechanism. A Dream application is expected to poll for user input frequently. The normal implementation of this behaviour is to call into the platform specific run-loop for a small duration of time to receive any pending events, and then fire them back into the provided Dream input handler.

During this time, any other platform specific mechanisms will have time to run, including application specific events such as video input processing.

3.3 Display Context

The basic Dream application includes a display context which is a generic interface to a platform specific mechanism for displaying OpenGL content. On a desktop computer this could be a window or a full screen, on a mobile device this is often the full screen.

Dream provides specific implementations for all supported platforms, including Mac OS X (Cocoa) and iOS (UIKit). In addition to this, Dream can be invoked as part of a natural application.

The display context will integrate with the host operating system in the most efficient way possible. In the case of Mac OS X, it will use a Core Video Display Link, and on iOS it uses a fixed frame timer. It currently does not support Display Link on iOS, but support for this will be added in the future and will not require changes to existing applications.

3.4 OpenGL Rendering

Basic geometric primitives can be rendered using the supporting Wireframe and Solid renderers. These provide simple rendering for OpenGL and OpenGL ES targets. More advanced rendering tasks are expected to be implemented directly by the application.

4 Sample Application

Finger Paint is a simple application which tracks a single marker and allows the user to draw on the plane defined by the marker using their finger on the touch screen. There are several non-trivial side-effects of this approach to drawing:

- The user can choose their distance from the marker either to draw a large outline or add small details.
- The user can view the marker, and thus image, from different perspectives.

Finger Paint is not a complex application to develop, but serves as a basis for developing and testing the cross-platform abstract interface to ARToolKit, and uses all the fundamental features of ARToolKit, including video input, marker tracking and geometric transformations.

Figure 3: Finger Paint running on an iPhone 3G. Four circles have been drawn already.

4.1 Cross Platform Support

Applications developed using the Dream+ARToolKit framework can execute on both Mac OS X (see figure 4) and iPhone devices (see figure 5) with very few changes.

Figure 4: Finger Paint application running on Mac OS X. Drawing is done using the track pad.

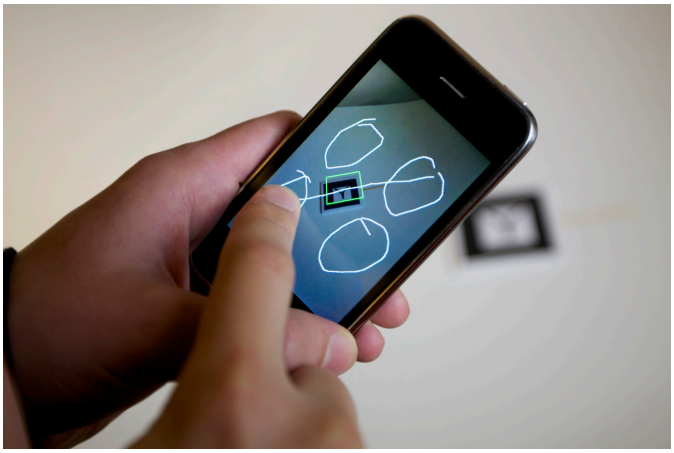

Figure 5: A line is drawn by dragging the finger across the screen.

The basic mechanism behind this is support for cross-compilation of all required dependencies and a unified presentation of associated libraries and headers. This is done using build scripts which provide all the necessary infrastructure for compiling and installing a given package.

Applications developed using Dream must be written in C++ and compiled once for each platform. This is the desired approach because there are many platform specific behaviours (i.e. code signing for iOS) which would be undesirable to unify. The code used to compile for each platform can essentially be identical.

5 Evaluation

5.1 Dream

The Dream framework has been successful in several projects thus far. It simplifies development of complex applications and provides standardised interfaces for reoccurring tasks in game development.

The ARToolKit wrapper works well and as reduces the complexity of a program using the ARToolKit library. The sample application was easy to develop and porting the code to the iOS platform did not represent a large challenge.

5.2 Finger Paint

Finger Paint has been evaluated by several users and has received positive feedback.

	Easy?	Intuitive?	Fun?
No			
Moderately	3	1	
Yes	2	4	5

Many users enjoyed drawing various kinds of pictures. One common criticism was that marker tracking is sometimes inaccurate and causes frustration.

6 Discussion

6.1 Cross-platform Abstractions

Cross-platform abstractions are exceedingly hard to get right. It is almost always necessary to try several different approaches to exposing the underlying functionality before settling on a balanced best-fit solution.

One approach to this style of development involves creating specific modules with minimal coupling. This reduces the cost of fixing/removing bad abstractions. Starting off with a simple interface and increasing exposure to underlying data as required ensures that abstractions are suitable for the majority of programs while not exposing unnecessary detail.

6.2 Consistent Behaviour

By design, the high level abstractions are implemented in terms of platform specific software stacks. Ensuring consistent and correct behaviour is difficult and time consuming. Low level platform structure can be completely different. Exposing platform specific behaviour (i.e. leaky abstraction) means that higher level components may be more complex to implement correctly, and in many cases very difficult to test in isolation.

Compilation and development workflow can be hugely different across platforms, to the point where completely different programming environments are used on different devices ensuring that all code must be written once for each platform. Some of these issues are insurmountable with the current framework.

6.3 Video Input

Video hardware quality is fairly poor in the majority of devices tested. Cameras were easily susceptible to motion blur and performed poorly in low light conditions.

One of the benefits of using an abstract interface for video input is that as hardware and software quality increase, these benefits will be transparently passed on to higher level applications.

6.4 Real-time Performance

Desktop class hardware is certainly powerful enough to perform the required processing, however mobile computers still lack the required processing power for real-time marker tracking.

This is primarily due to low level processor performance. Many mobile devices have poor floating point performance, and limited memory capacity and bandwidth. As processors improve, application performance will naturally improve along with it.

Another option to improve performance would be to profile the low level algorithms and look for areas that could be improved. Having a high level abstract interface means that low level algorithms can be improved and many high level applications will benefit.

6.5 ARToolKit

ARToolKit creates several challenges when developing abstract interfaces.

- Many data types can only be provided from a named file, as opposed to being loaded from a memory buffer (i.e. pattern files). It would be convenient to use a standard file format (i.e. PNG) and load data from memory buffers rather than file paths.
- There is a high coupling between platform specific rendering functions and the marker tracking mathematics. There are various different libraries provided to deal with this issue, such as *ARgsub*, *ARgsub_lite*, but they don't make it easy to create clean abstractions.
- There is also a high level of coupling between the video input and marker detection functions. While this is very convenient, it is not possible to manage video frame buffers without explicit copying if required. This feature was not required for the current implementation, but it might be a requirement in the future, in which case it could become a performance liability.
- ARToolKit2 depends on global state. This makes it impossible to cleanly abstract into a general set of object-oriented structures, and may increase the complexity of distributing marker detection over multiple processors. Fortunately ARToolKit4 has removed many of these global dependencies.

6.6 Software Development

The main benefit of Dream and the platform-independent approach is that software can be developed once and run on a multitude of platforms. The general equation is:

$$O(N + M) \qquad vs \qquad O(N \times M)$$

where N is the number of applications and M is the number of platforms. By decoupling the underlying platform from the application, we reduce the cost of development. Also, because Dream provides a significant existing foundation for interactive graphical applications, overall development time will be reduced.

6.7 Further Work

The existing video interface provides cross-platform support for ARToolKit integration. However, there are still some issues to be addressed. Video frames may need to be rotated and flipped to be displayed correctly on the screen. This may depend on the kind of camera and its orientation. This is currently handled in a platform-specific manor.

ARToolKit is not fast on mobile class hardware. It may be possible to optimise it to improve performance, and also adapt the tracking algorithms to better suit the quality of the cameras available on mobile devices.

Further work is required to enhance the Dream build system. Some parts of it currently depend on Xcode. However, it should be relatively easy to expand upon the existing build cross-platform build system to remove this dependency. This would improve cross-platform support.

7 Conclusion

Integrating Dream and ARToolKit has been a successful research project and resulted in the ability to create augmented reality applications with minimal platform dependency. A sample application – Finger Paint – was developed and can run on both Mac OS X and iOS. It has been used to investigate the benefits of a high level approach, and as part of this study, further integration challenges have been revealed.

Developing mobile applications successfully involves additional concerns beyond that of regular desktop computer software engineering. This is due to the many differences in the hardware and software platforms and the specific requirements of modern augmented reality applications. In many cases issues can be minimised by carefully developing cross-platform interfaces and supporting tools, however there are also inherent limitations in hardware which cannot be overcome.

Mobile computers are an important technology platform for the growth of new ideas in augmented reality. Understanding both the limitations and benefits of mobile computers can help software developers produce better applications, and better foundational software platforms can reduce the cost of software development.

Reducing the difficulties associated with mobile software development will also hopefully encourage the development more sophisticated applications, since software developers have more time to focus on higher level features and integration challenges rather than low level algorithmic and structural challenges.

Mobile computers will continue to increase in power and capability; however there will always be differences between a computer that fits in your hand and a computer that sits on your desk. History has shown us that even though this gap will continue to shrink, it is likely there will always be important considerations to keep in mind when developing mobile augmented reality applications.

References

[1] Gartner. Mobile phone sales soar, 2010. [Online; accessed Sep-2010]. Available from: http://www.physorg.com/news193487624.html.

[2] Samuel Williams. Dream. Available from: http://www.oriontransfer.co.nz/research/dream.

[3] Studierstube augmented reality project, 2010. [Online; accessed Sep-2010]. Available from: http://studierstube.icg.tu-graz.ac.at/.

[4] Daniel Wagner and Dieter Schmalstieg. Making augmented reality practical on mobile phones, part 1. *IEEE Comput. Graph. Appl.*, 29(3):12–15, 2009.

[5] Daniel Wagner and Dieter Schmalstieg. Making augmented reality practical on mobile phones, part 2. *IEEE Comput. Graph. Appl.*, 29(4):6–9, 2009.

[6] Oolong engine. Available from: http://code.google.com/p/oolongengine/.

[7] Sio2 engine. Available from: http://sio2interactive.com/.

[8] Impact for ios. Available from: http://www.phoboslab.org/log/2010/10/impact-for-ios.