Mobile AR Game Engine

Using iPhone and Mac OS X.

Samuel Williams

Project Outline

- Integrate an open source game engine on the iPhone or Android operating system with the mobile version of ARToolKit to produce a library for developing mobile AR applications.
- Develop a simple sample application to demonstrate how the game engine can be used.

Ideal Solution

- Game engines that are cross platform simplify application development.
- Underlying hardware and software interfaces are abstracted.

Main Problems

- iPhone has its own runloop which is used by core system functionality, including camera.
- iPhone exposes several compilation variants including x86 (simulator) ARMv6 and ARMv6.
- iPhone has a different set of high level frameworks compared to Mac OS X.

How were these problems solved?

Runloop

- Event handling calls into main runloop and this is done once per frame.
- This receives both user input from the device, and updates external dependencies such as camera.
- Abstract event pipeline is compatible with a wide range of user interfaces (Mac OS X, X11, iPhone, etc)

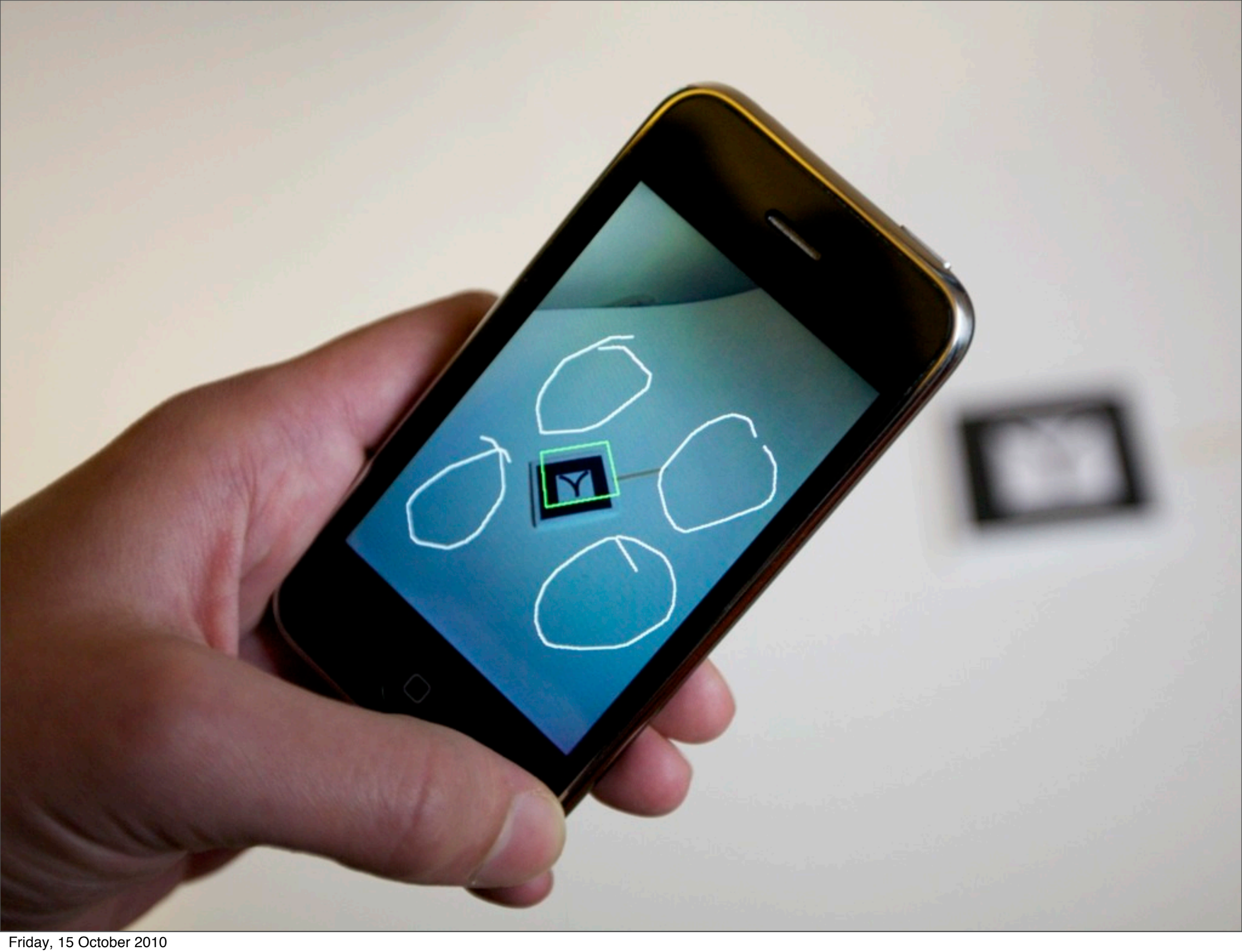
Compilation

- A specific build system written in Ruby allows source code to be compiled for multiple platforms easily.
- Cross platform compilation of boost, libpng, libjpeg, etc.
- Integration with ARToolKit both for Mac OS X and iPhone using different libraries.

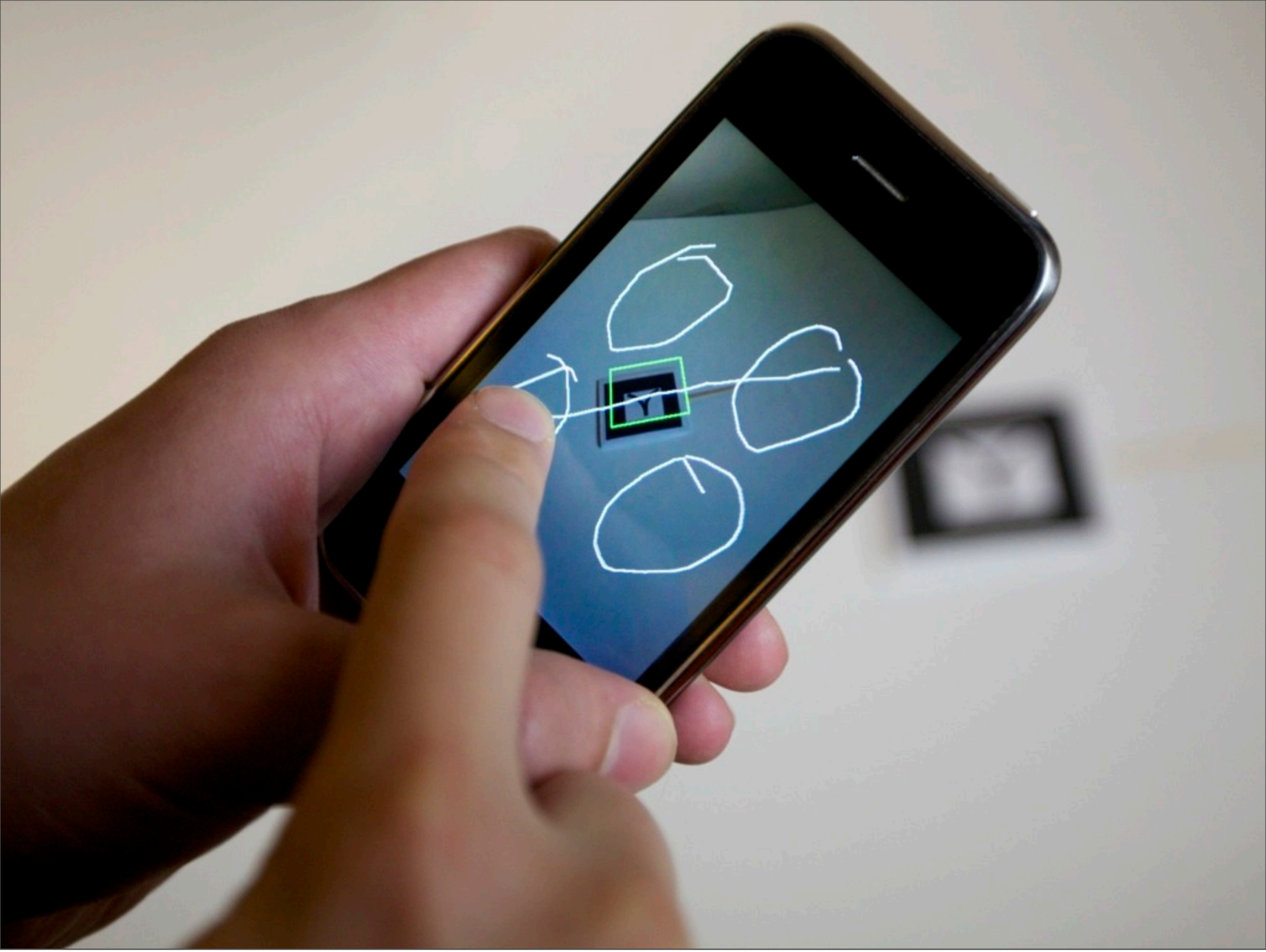
Abstract Interfaces

- Dream framework is a free software game development framework in C++.
- Provides high level interfaces for most major parts of a game engine.
- Hides low level details as much as is reasonable and useful.

Finger Paint



Friday, 15 October 2010



Why is this interesting?

This application was developed almost entirely on Mac OS X using the Dream framework. The same source code was then compiled for iPhone. It was far simpler to debug problems on Mac OS X using a full and modern development environment.

Feedback on Finger Paint

- Generally people enjoyed it and had fun drawing things.
- Rotation and transformation of the drawing could produce interesting results.
- Most people ranked it as being fun, easy and intuitive.

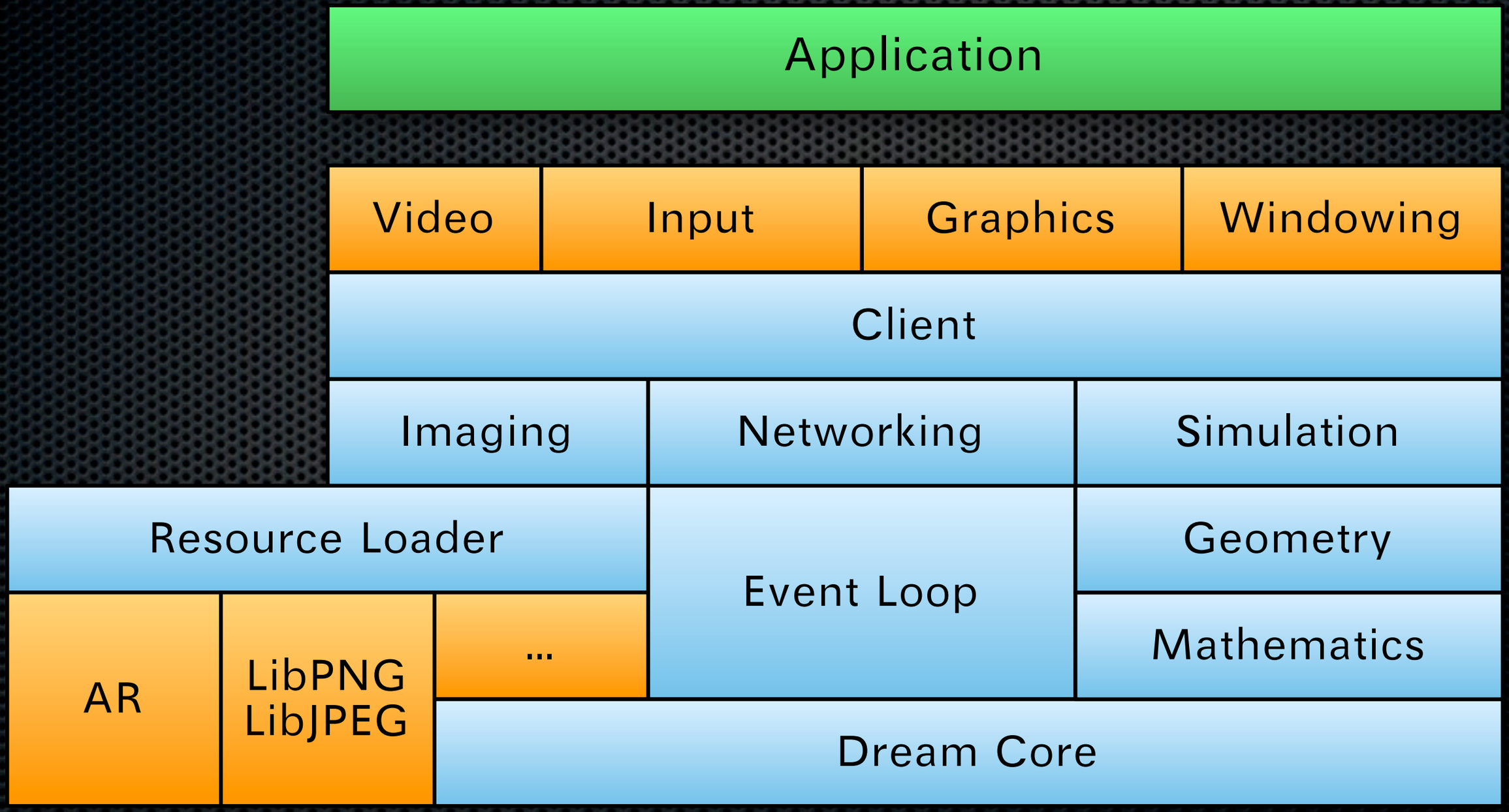
A small qualitative study was done with several people.

Ultimately...

- We are trying to reduce the workload to produce new applications and support new platforms.
- A major problem in the current device ecology.

What is the point of this project? Why make a library? Why provide integration?

Dream Framework



Green areas show application code. Blue areas show Dream framework code. Orange areas show platform dependent abstract interfaces.

Video Integration

- Video on iPhone and Mac OS X is very different.
- iPhone video hooks into the native run-loop.
- Exposed through display context using the input delegate.
- Supports both GPL ARToolKit 2, and ARToolKit Pro.

There are completely different frameworks for accessing video frames. There are two different methods depending on what iPhone version you target.

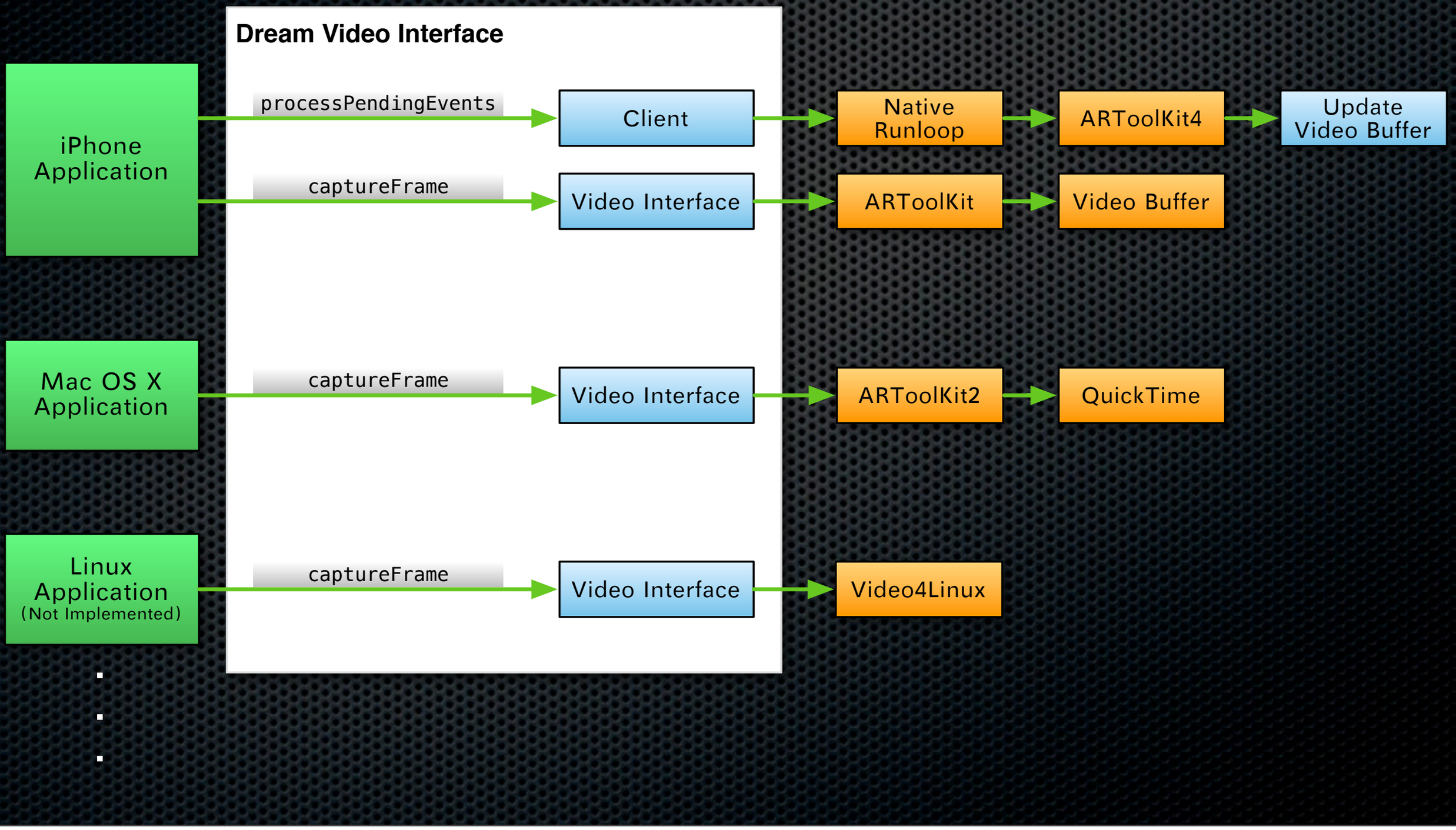
Implementation of run-loop integration (CoreFoundation)

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

   SInt32 result;

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

   g_inputHandler = NULL;
}
```

Abstract cross-platform cross-library interface (Dream)

```
// Read frame from camera.
REF(IPixelBuffer) videoFrame = m_videoInput->captureFrame();

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

Cross-platform Video



Lessons Learned

- Cross-platform abstraction is very hard to get right.
- You might need to try out several different approaches before you find the right level of abstraction.
- A modular approach with minimal coupling seems to work well since it reduces the cost of removing bad abstractions.

Lessons Learned

- By nature, the high level abstraction sits on top of the platform-specific software stack: ensuring consistent and correct behaviour is not easy.
- Low level program flow can be completely different between platforms.
- Compilation and workflow can be hugely different across platforms.

Lessons Learned

- Video hardware quality is still fairly poor.
- Majority of devices still not powerful enough for “real time” marker tracking using ARToolKit.

Lessons Learned

- ARToolKit has many impedance mismatches in its abstract interface.
  - Many data types can only be provided by file names (rather than data buffers).
  - Integration of mathematics drawing and marker tracking very dependent on native graphics implementation.
  - Dependence on global state.

Main Benefits

- Develop the software once, compile it for multiple platforms: O(N+M) where N is the number of applications and M is the number of platforms.
- Reduce the cost of application development by providing a significant existing foundation.

Further Work Required

- Better abstractions and interfaces.
  - Handle rotated and flipped video input automatically.
- Major performance problems with ARToolKit on older hardware.
  - Recognition is highly dependent on light.
  - Performance of detection is about 10 frames per second.

Questions?