## .kkapture: Guided tour

Fabian 'ryg' Giesen

Farbrausch

Fabian 'ryg' Giesen .kkapture: Guided tour Farbrausch

Basic principles

### Outline

### Introduction

### **Basic principles**

The main idea Intercepting API calls

### kkapture piece by piece

Video encoders Video APIs Audio

### Overview

- Learn what kkapture can do, how it does it...
- ... and how to teach it new tricks if you need/want to.
- How to make demos kkapture-friendly?
- Get you to fix kkapture yourself if it barfs on your demo.
  - But send back patches, please :)
- Plus some anecdotes...

### But first...

Let's debunk some common misconceptions:

- kkapture is not D3D only.
- kkapture is **not** a screengrabber.
- It doesn't "hack" your system, either.
  - Everything kkapture does is local to target app.
- > You can let kkapture run in the background.
  - But: Consumes lots of CPU power and I/O bandwidth.
  - Also, a lot of demos stop running when they lose focus.
- Directly encoding to XVid/H.264/etc. from kkapture is a bad idea
  - Use fast, lossless codec for kkapturing. (HuffYUV, LagArith etc.)
  - Transcode to actual target format later. (VirtualDub!)

> All graphics APIs (that we're interested in) are double-buffered: while (!done) { updateStuff(); renderStuff(); // to (invisible) back buffer swapBuffers(); // make back buffer visible }

### The main idea

All graphics APIs (that we're interested in) are double-buffered: while (!done) { updateStuff();

```
renderStuff(); // to (invisible) back buffer
```

```
swapBuffers(); // make back buffer visible
```

- }
- Intercept that one call and we know when to grab the image.
- Different function for different APIs, so need to handle them separately.
- > The twist: Make time "stand still" between successive calls.
  - Simulate "infinitely fast" CPU that always waits for graphics to finish rendering.
  - Also pretend that rendering takes a fixed time.
- Why?
  - Rendering videos is expensive, especially at high resolution and framerate.
  - ► Removing "real-time" from the equation makes everything easier.

### How to build a time machine

- Need to control time as visible to app.
- Tons of possible time sources:
  - Direct: GetTickCount(), QueryPerformanceCounter() etc.
  - Indirect: Sleep(), WaitForSingleObject() etc.
  - Event-based: timeSetEvent(), SetTimer()
  - Sound: Current play position (emulate sound card!)
  - CPU: RDTSC (hard to do; kkapture ignores this)
- Intercept them all, make them report consistent values.

### How to build a time machine

- Need to control time as visible to app.
- Tons of possible time sources:
  - Direct: GetTickCount(), QueryPerformanceCounter() etc.
  - Indirect: Sleep(), WaitForSingleObject() etc.
  - Event-based: timeSetEvent(), SetTimer()
  - Sound: Current play position (emulate sound card!)
  - CPU: RDTSC (hard to do; kkapture ignores this)
- Intercept them all, make them report consistent values.
  - Congratulations, you now control time.
- That's all we need to do—now how do we do it?

## Intercepting API calls

- If we had the source code to the demo, this would be easy.
- Link with a library that replaces system calls:

```
BOOL SwapBuffers(HDC hdc) {
  grabCurrentFrame();
  return Real_SwapBuffers(hdc);
}
DWORD GetTickCount(void) {
  return start + currentFrame * msPerFrame;
}
.
```

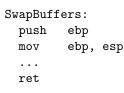
and so on.

- But we don't, so we have to do this using the binary only.
- On Linux, there's LD\_PRELOAD, but this is Windows, so we have to do it ourselves.

### **Binary Instrumentation**

- Just patch the target program.
- Calls are hard to find:
  - Just search for matching byte sequence and change code?
    - What if it's a false positive?
  - ► Different opcodes: CALL, JMP (short and near) etc.
  - Indirect calls, jump tables, ...
  - Complete program flow analysis?!
- $\blacktriangleright$   $\Rightarrow$  Patch the **destination**, not the call site.

## **Binary Instrumentation (2)**



Interception:

- 1. Copy first few opcodes of target function to "trampoline" function.
- 2. Write hook function.
- 3. Overwrite start of original function with jump to hook function.
- 4. Hook can continue real function via "trampoline".

kkapture uses a library (Detours) for this.

## **Binary Instrumentation (2)**

```
SwapBuffers:

push ebp

mov ebp, esp

...

ret
```

```
RealSwapBuffers:
```

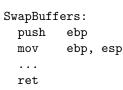
push	ebp
mov	ebp, esp
•••	
jmp	SwapBuffers+5

Interception:

- 1. Copy first few opcodes of target function to "trampoline" function.
- 2. Write hook function.
- 3. Overwrite start of original function with jump to hook function.
- 4. Hook can continue real function via "trampoline".

kkapture uses a library (Detours) for this.

## **Binary Instrumentation (2)**



```
RealSwapBuffers:
```

push	ebp
mov	ebp, esp
 jmp	SwapBuffers+5

### MySwapBuffers:

Interception:

- 1. Copy first few opcodes of target function to "trampoline" function.
- 2. Write hook function.
- 3. Overwrite start of original function with jump to hook function.
- 4. Hook can continue real function via "trampoline".

kkapture uses a library (Detours) for this.

. . .

## **Binary Instrumentation (2)**

### SwapBuffers:

MySwapBuffers

### RealSwapBuffers:

push	ebp
mov	ebp, esp
 jmp	SwapBuffers+5

MySwapBuffers:

Interception:

- 1. Copy first few opcodes of target function to "trampoline" function.
- 2. Write hook function.
- 3. Overwrite start of original function with jump to hook function.
- 4. Hook can continue real function via "trampoline".

kkapture uses a library (Detours) for this.

. . .

## **Binary Instrumentation (2)**

### SwapBuffers:

jmp	MySwapBuffers
• • •	
• • •	
ret	

### RealSwapBuffers:

push	ebp
mov	ebp, esp
 jmp	SwapBuffers+5

MySwapBuffers:



### Interception:

- 1. Copy first few opcodes of target function to "trampoline" function.
- 2. Write hook function.
- 3. Overwrite start of original function with jump to hook function.
- 4. Hook can continue real function via "trampoline".

kkapture uses a library (Detours) for this.

. . .

## Interception odds & ends

- Important: You can only call/jump to functions in the address space of your target process.
- $\blacktriangleright$   $\Rightarrow$  Need to get the code in there somehow.
- Process:
  - 1. Start target process as suspended (i.e. not running).
  - 2. Allocate some memory in target code, put our init code there.
  - 3. Detour startup code (just like you would any other function).
  - 4. Init code loads kkapturedll.dll (contains all our code).
  - 5. kkapturedll startup code sets up interception of everything.
- For virtual functions (e.g. COM interfaces): Actual address of function is in virtual function table (a per-class jump table), just get address to patch from there.

#### Video encoders

### Video encoders

```
Relatively simple interface:
  class VideoEncoder {
  public:
    virtual ~VideoEncoder();
    virtual void SetSize(...);
    virtual void WriteFrame(...);
    virtual void SetAudioFormat(...);
    virtual void GetAudioFormat(...);
    virtual void WriteAudioFrame(...);
  };
```

Global VideoEncoder\* encoder is pointer to encoder to use.

#### Video encoders

### Video encoders

```
Relatively simple interface:
  class VideoEncoder {
  public:
    virtual ~VideoEncoder():
    virtual void SetSize(...):
    virtual void WriteFrame(...);
    virtual void SetAudioFormat(...);
    virtual void GetAudioFormat(...);
    virtual void WriteAudioFrame(...);
```

```
};
```

- ► Global VideoEncoder\* encoder is pointer to encoder to use.
- Actual implementations: **boring** (and/or tedious, e.g. DShow).
- Moving on...

## Video APIs (1)

- Example here: OpenGL (others are similar).
- Basic flow as outlined above, need to intercept:
  - ChangeDisplaySettingsEx (video mode changes).
  - wglCreateContext and variants.
  - wglMakeCurrent
  - SwapBuffers and variants.
- Note: ChangeDisplaySettings calls Ex variant internally, so only intercept Ex!
- ... Redirection affects all code running in target process, including System DLLs – so careful, there can be side effects!

## Video APIs (2)

### Why track rendering contexts?

- Users call SwapBuffers with HDC, but need to know which rendering context belongs to that DC.
- Make it active if it's not already.

## Video APIs (2)

### Why track rendering contexts?

- Users call SwapBuffers with HDC, but need to know which rendering context belongs to that DC.
- Make it active if it's not already.
- SwapBuffers in detail:
  - Grab actual frame data (via glReadPixels).
  - Encode it: encoder->WriteFrame(captureData);
  - Mark frame as completed: nextFrame();

## Video APIs (2)

### Why track rendering contexts?

- Users call SwapBuffers with HDC, but need to know which rendering context belongs to that DC.
- Make it active if it's not already.
- SwapBuffers in detail:
  - Grab actual frame data (via glReadPixels).
  - Encode it: encoder->WriteFrame(captureData);
  - Mark frame as completed: nextFrame();
- Other APIs work similarly.

Audio

Basic principles

### Almost the end...

# **Questions?**

Fabian 'ryg' Giesen

.kkapture: Guided tour

Farbrausch

	Basic principles	
Audio		

### ryg@theprodukkt.com http://www.farbrausch.de/~fg/kkapture

### Some links

Detours http://research.microsoft.com/sn/detours HuffYUV just Google it :) LagArith http://lags.leetcode.net/codec.html VirtualDub http://www.virtualdub.org x264 http://x264.nl

kkapture piece by piece