# 대화면 FPS 게임을 위한 새로운 레이저기반 입력장치

한녹손 (1), 김성환 (1), 박인규 (2)
서울시립대학교 컴퓨터과학부 (1), D&C Applications Inc. (2)
e-mail : han.ngoc.son@gmail.com, swkim7@uos.ac.kr, icpark@uos.ac.kr

# New Input Device for Large Screen First Person Shooter Games

Ngoc–Son Han (1), Seong-Whan Kim (1), and In-Kyu Park (2)
Dept. of Computer Science, University of Seoul (1)
D&C Applications Inc. (2)

**Abstract**

*In this paper, we present a new game interface design for First Person Shooters (FPS). Previously, FPSs on computer are commonly played using keyboard/mouse or joystick along with PC display. We improve the communication environment between player and game world by means of new control system including large screen, laser gun, and directional device, which create a real life-like space for players. Because traditional display for FPS uses CRT, it cannot support large screen display due to limitation of CRT technology. We designed and implemented a new input device using laser recognizable display. We implemented a new FPS based on Quake III that is in accordance with the new devices. Results suggest that the combined interface creates a method which helps beginners to enjoy playing a FPS immediately and gives experienced players a new gaming experience.*

## 1. Introduction

A first-person shooter (FPS) is a video game that renders the game world from the visual perspective of the player character and tests the player's skill in aiming guns or other projectile weapons. All FPS feature the core gameplay elements of movement and shooting, but many variations exist, with different titles emphasizing certain aspects of the gameplay. Most modern first-person shooters on the PC utilize a combination of the WASD keys of the keyboard and mouse as a means of controlling the game (commonly referred to as "WASD/Mouse"). One hand uses the mouse, which is used for free look (also known as mouse look), aiming and turning the player's axis. On the keyboard, the arrow keys (or other keys arranged in the same manner, such as WASD, ESDF or IJKL) provide digital movement forwards, backwards, and sidestepping (often known as "strafing" among players) left and right. Usually these buttons make the player run, and a nearby button must be pressed in order to walk.


*Fig. 1. First Person Shooter.*

We improve the communication environment between player and game world by means of new control system including large screen, laser gun, and directional device, which create a real life-like space for players. Because traditional display for FPS uses CRT, it cannot support large screen display due to limitation of CRT technology. We designed and implemented a new input device using laser recognizable display. We implemented a new FPS based on Quake III that is in accordance with the new devices. Results suggest that the combined interface creates a method which helps beginners to enjoy playing a FPS immediately and gives experienced players a new gaming experience.

We review related technologies for FPSs in section 2, and propose and show experimental results of our new input device in section 3, section 4 and section 5. We conclude in section 6.

## 2. Related Works

A first-person shooter (FPS) is a video game that renders the game world from the visual perspective of the player character and tests the player's skill in aiming guns or other projectile weapons. Most modern first-person shooters on the PC utilize a combination of the WASD keys of the keyboard and mouse as a means of controlling the game (commonly referred to as "WASD/Mouse").

Recently a new gaming device and interaction method for FPS based on ChairIO has been developed by University of Hamburg. ChairIO is based on a stool and is similar to a joystick, but controlled by the user's body motion. The ChairIO interaction is augmented by a game console gun to form a new interaction method for FPSs. An initial evaluation compares several ChairIO and gun

based interaction methods with traditional keyboard and joystick controls. They developed a gun in three degrees of freedom using InertiaCube2 a motion tracking product using Virtual Reality (VR) from InterSense Inc. This tracking device provides an absolute rotation in three axes with real-time update frequency and minimal drift.

*Fig. 2. FPS using ChairIO and InertiaCube2.*

Because traditional display for FPS uses CRT, it cannot support large screen display due to limitation of CRT technology. We designed and implemented a new input device using laser recognizable display. We implemented a new FPS based on Quake III that is in accordance with new devices. The primary advantage of CRT monitors is their color rendering. The contrast ratios and depths of colors displayed were much greater with CRT monitors than other type of monitors. The other advantage that CRT monitors is the ability to easily scale to various resolutions. This is referred to as multisync by the industry. By adjusting the electron beam in the tube, the screen can easily be adjusted downward to lower resolutions while keeping the picture clarity intact. While these two items may play an important role for CRT monitors, there are disadvantages. The biggest of these are the size and weight of the tubes. An equivalent sized LCD monitor for example is upwards of 80% smaller in size and weight compared to a CRT tube. The other major drawback deals with the power consumption. The energy needed for the electron beam means that the monitors consumer and generate a lot heat. Cons of large CRT display:

- Very heavy and big.
- Use large amounts of energy.
- Generate excess heat.

## 3. LaserTouch: new input device for large screen FPS

Our new control system is different from University of Hamburg's which has to deal with the problem of accuracy. Our approach comes from interplay between the *large screen* and a specially designed laser gun. The Laser beam coming from the gun draws a marker on the screen and its position is recognized and is inputted to the computer with other trigger information.

There are three components in our new control system, las*er gun*, *large screen* and *directional device*. The Laser gun is installed away from the large screen, it can fire laser beam that will reach the screen and locate the position of target for the player. In addition, laser gun contains a set of buttons that deal with activities involved with weapon and ammunition such as switching weapon, firing. The Large screen is a special one made of optical sensors with capabilities of rendering image and receiving signals from laser gun.

The Directional device in the form of a chair is developed similar to the ChairIO, for use as a joystick; and operation techniques allow the game characters to move around. Drivers for new devices are written under the scope of Window Driver Foundation (WDF), Windows Display Driver Model for large screen and Kernel-Mode Driver Framework (KMDF) for the laser gun and the screen. On top of that new control system, we develop a FPS called QuakeS based on Quake III whose source code was released under GPL license. A new module using DirectInput Windows API for collecting input from user via game controllers for input control is added to interact with new devices.

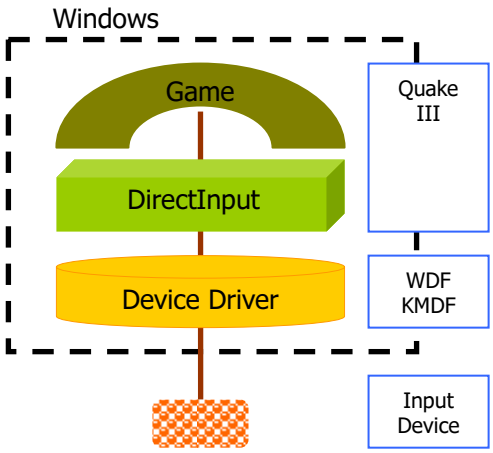


*Fig. 3. FPS with New Input Devices Architecture.*

Fig. 3 shows the architecture of FPS with new input devices. At the top of the model, QuakeS game is running under Windows operating system providing environment for players to join and play with core game. Players first-person character inside the game use input devices to move and do action by changing state of the input devices such as moving laser gun up and down, right and left. In order to interact with input devices, QuakeS uses DirectInput library of DirectX to invoke devices routine functions that detect the changes of input devices. Such routines are created by Device Drivers that are built on WDF.

## 4. Device Driver Implementation for LaserTouch

Plug-n-Play (PnP) is a technology that allows for the hardware on the computer to be changed dynamically, and the PnP software will automatically detect changes, and allocate important system resources. PnP get's it's own root driver, that communicates closely with the Root bus driver, to keep track of the devices in your system. Each Plug and Play device must be uniquely identified, state the services it provides and resources it requires, identify the driver that supports it, and allow software to configure it. The PnP Manager loads at most one function driver for a device. A function driver can service one or more devices.

A driver typically communicates with the device through the computer bus or communications subsystem to which the hardware is connected. When a calling program invokes a routine in the driver, the driver issues commands to the device. Once the device sends data back to the driver, the driver may invoke routines in the original calling program. Drivers are hardware-dependent and operating-system-specific. They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface. Windows device drivers generally come in 2 flavors: **Virtual Device Drivers** (VxD) and **Windows Driver Model** (WDM). VXD style drivers are older, and are less compatible, while WDM drivers are supposed to be fully code-compatible all the way back to Windows 95.

Device Drivers operate in kernel mode so writing, testing, and debugging drivers can be a tricky task. Drivers should always be well tested before they are installed. Since device drivers do not operate in user mode, the user mode libraries (kernel32.dll, user32.dll, wingdi.dll, msvcrt.dll) are not available to a device driver. Instead, a device driver must link directly to the Native API functions in ntdll.dll, and even lower-level functions in the kernel itself. Device drivers are typically written in C, using the Driver Development Kit (DDK). There are functional and object-oriented ways to program drivers, depending on the language chosen to write in. It is generally not possible to program a driver in VB. Because drivers operate in kernel mode, there are no restrictions on the actions that a driver may take. A driver may read and write to protected areas of memory, it may access I/O ports directly, and can generally do all sorts of very powerful things. This power makes drivers exceptionally capable of crashing an otherwise stable system.

The Windows platform DDK comes with header files, library files, and a command-line compiler that can be used to write device drivers in C or C++. There is no graphical interface to the DDK compiler. The Windows Driver Kit (WDK) is a fully integrated driver development system for the Microsoft Windows family of operating systems. It combines the Windows DDK and Driver Test Manager (DTM), and also provides tests that Microsoft uses to test the stability and reliability of the Windows operating system.

We develop a driver for new input device (new input device here is used to refer the combination between the *large screen* and the *laser gun*) using Kernel-Mode Driver Framework. KMDF implements the fundamental features required for kernel-mode drivers, including: Plug and Play and power management, I/O queues, Direct memory access (DMA), Windows management instrumentation (WMI), and Synchronization. KMDF defines an object-based programming model in which object types represent common driver constructs. Each object exports methods (functions) and properties (data) that drivers can access and is associated with object-specific events, which drivers can support by providing event callbacks. The objects themselves are opaque to the driver.

| **Input Device** | |
|---|---|
| *Operating System* | Windows family |
| *Driver Model* | Windows Driver Foundation (WDF) |
| *Input Control* | DirectInput |
| *Tools* | Windows Driver Kit<br>Visual Studio .NET |

*Fig. 4. Choice for New Input Device.*

KMDF creates some objects on behalf of the driver, and the driver creates others depending on its specific requirements. The driver also provides callbacks for the events for which the KMDF defaults do not suit its device and calls methods on the object to get and set properties and perform any additional actions. Consequently, a KMDF driver is essentially a DriverEntry routine, a set of callback functions that perform device-specific tasks, and whatever utility functions the driver implementation requires. New device driver uses the WDF defaults includes the following functions:

- A **DriverEntry** routine, which creates the driver object.
- An **EvtDriverDeviceAdd** event callback, which creates the device object, a device interface, and a default I/O queue.
- I/O callback functions for read, write, and device I/O control requests.
-

The **DriverEntry** routine is the first driver function called when the driver is loaded. The KMDF DriverEntry routine has prototype: NTSTATUS DriverEntry(

```
IN PDRIVER_OBJECT    DriverObject,
IN PUNICODE_STRING   RegistryPath
);
```

The DriverEntry routine performs the following tasks:

- Creating a driver object (WDFDRIVER), which represents the loaded instance of the driver in memory. In effect, creating this object “registers” the driver with KMDF.
- Registering the driver’s EvtDriverDeviceAdd callback. KMDF calls this function during device enumeration.
- Optionally initializing event tracing for the driver.
- Optionally allocating resources that are required on a driver-wide (rather than per-device) basis.

Our driver supporting Plug and Play has an EvtDriverDeviceAdd callback function, which is called each time the system enumerates a device that belongs to

the driver. This callback performs actions required at device enumeration, such as the following:

- Creating and initializing a device object (WDFDEVICE) and corresponding context areas.
- Setting entry points for the driver's Plug and Play and power management callbacks.
- Creating a device interface.
- Configuring and creating one or more I/O queues.
- Creating an interrupt object, if the device controls physical hardware that generates interrupts.

The EvtDriverDeviceAdd function is called with two parameters: a handle to the WDFDRIVER object that the driver created during DriverEntry and a handle to a WDFDEVICE_INIT object.Day

## 5. Quake III Modification for LaserTouch

We use Open Source FPS Quake III to implement new control device. **id Software** released the source code for Quake III under the GNU General Public License. Quake 3 is entirely written in C. Quake 3 solution is divided into 8 smaller modules (8 projects) including **botlib**, **cgame**, **game**, **q3_ui**, **quake3**, **renderer**, **splines** and **ui.**

DrectIntput module in Quake III allows developers to deal with external controllers. It uses Microsoft input library DirectInput (a part of DirectX) supporting keyboard, mouse, joystick, and other game controllers. DirectInput enables an application to retrieve data from input devices even when the application is in the background. It also provides full support for any type of input device. Through action mapping, applications can retrieve input data without needing to know what kind of device is being used to generate it. Quake III DirectInput functions are wrapped inside win_input.c under project quake3. It originally provides full implementation for mouse and joystick that includes some basic steps.

1. Create the DirectInput object
2. Create a DirectInputDevice object for each device you want to use
3. Set up and acquire the device.
4. Retrieve and act on the data.

We implement the following functions to deal with new input device. These functions initial the communication between game components with out new input device, register with DirectInput and get an IDirectInput to play with, obtain an interface to the system mouse device, set the data format to "mouse format", and set the cooperativity level. By which the new input device operation will be similar to system mouse device that the core Quake III game well supports.

```
/*========================
  IN_InitDIControl
  ========================*/
  qboolean IN_InitDIControl( void ) {}
  /*==========================
  IN_ShutdownDIControl
  ==========================*/
  void IN_ShutdownDIControl( void ) {}
  /*==========================
  IN_ActivateDIControl
  ==========================*/
  void IN_ActivateDIControl( void ) {}
  /*==========================
  IN_DeactivateDIControl
  ==========================*/
  void IN_DeactivateDIControl( void ) {}
  /*====================
  IN_DIControl
  ====================*/
  void IN_DIControl( int *mx, int *my ) {}
```

*Fig. 5. New Input Device Implementation*

## 6. Conclusion

In this paper, we have presented new design for control system of FPSs. We coved several related issues ranging from hardware design to software design that help building and deploying new devices for a practical game. Three components of new control system including *laser gun*, *large screen* and *directional device* were designed and manufactured with supported Windows Drivers that compatible with Windows Application. Device drivers were developed with Windows Driver Kit and using WDF model. On the game side, input control module inside QuakeS used DirectInput library to deal with input signals.

Besides that the new control system creates fun and natural environment for players. And despite of the absence of some activities within the game, but almost skilled players find new control system worth experiencing.

## 7. References

[1] S. Beckhaus, K. J. Blom, and M. Haringer, *A new gaming device and interaction method for a First-Person-Shooter*, In Computer Science and Magic 2005, GC Developer Science Track, Leipzig, Germany, 2005.

[2] Microsoft Windows Hardware Developer Central, http://www.microsoft.com/whdc.

[3] Microsoft, *Architecture of the Kernel-Mode Driver Framework*, Microsoft Corporation, 2006.