

СОДЕРЖАНИЕ

Введение в WideGlance	03
Торговые марки, патенты	03
Глоссарий	03
Технические детали. Аппаратно-программный комплекс для определения координат пользователя в пространстве ruCap CMU-3	05
Аппаратная часть	05
Принцип работы	05
Программная часть	06
Использование	06
Отличия от других устройств ввода. Совместимость	07
Описание функций драйвера ruCap CMU-3.....	08
Перечень экспортируемых функций	08
wg_result_t GetSignature(LPRUCAPSIGNATUREDATA pSignature)	08
wg_result_t GetDriverVersion(int& nVersion)	08
wg_result_t SetSoundSpeed(float Speed)	09
float GetSoundSpeed()	09
wg_result_t SetMask(char Mask)	09
char GetMask()	09
wg_result_t SetFilter(int CommonFilter,int SelectedFilter,int ApproxFactor)	09
wg_result_t GetFilter(int &CommonFilter,int &SelectedFilter,int&ApproxFactor)	10
wg_result_t SetMonitorWidth(float W)	10
float GetMonitorWidth()	10
int GetHeadPos(float& x,float&y,float& z)	10
wg_result_t SetEyeOffset(int nModelCode,float x,float y,float z)	10
wg_result_t GetEyeOffset(int& nModelCode,float& x,float&y,float& z)	10
wg_result_t SetScreenCenterPos(float x,float y,float z)	11
wg_result_t GetScreenCenterPos(int& nModelCode,float& x,float&y,float& z)	11
Программные компоненты SDK	11
Использование выдаваемых драйвером данных. Первичная обработка данных	12
Загрузка драйвера	12
Опрос устройства	15
Коррекция положения глаз. Модели	15
Ортогональная модель	15
Использование сферической системы координат	16
Анатомические модели	16
Сидящий человек	16
Стоящий человек	16
Усовершенствования анатомической модели	16
Стереоиображение	17
Обработка данных	17
Использование технологии WideGlance в игровом движке	20
Действия во время серверного фрейма	20
Физика	20
Зачем потребовались изменения в физике	20
Возникшие сложности	20
Тонкости внедрения неинерционного смещения	21
Изменения в физике	21
Игровая логика	22
Оружие	22

Перемещение модели оружия	22
Перемещение точки вылета снарядов	22
Изменение направления вылета снарядов	22
Перемещение или неперемещение прицела	22
Используемые объекты	22
Обработка ввода	23
Перемещение	23
Взаимодействие с объектами среды. Подбирание предметов	23
Наведение на объекты	23
Прочие объекты	23
Действия при подготовке к рендеру и во время рендера	24
Перемещение коллизионного объема (физического тела) синхронно с головой	29
Первый подход – применение силы	29
Второй подход – неинерционное движение	29
Программирование оружия	34
Положение прицела на экране. Режимы наведения оружия	34
Направление вылета снарядов	34
Точка вылета снарядов	34
Положение модели оружия	34
Проблема наведения оружия (конфликт устройств ввода)	35
Снайперский режим	35
Скорость полета снарядов при использовании неинерционного перемещения	35
Этапы встраивания технологии в игровой движок	36
Шаги встраивания	36
Шаги, которые могут еще потребоваться	37
Примеры	38
Игра «Жор-тур» (3D-движок “Shine”)	38
Меню настройки	38
Консольные переменные и команды	40
vr_WideGlanceEnable	40
vr_WideGlance_XPos, vr_WideGlance_YPos, vr_WideGlance_Zpos	40
vr_WideGlance_OriginalXPos, vr_WideGlance_OriginalYPos, vr_WideGlance_OriginalZPos	40
vr_WideGlanceKeyBoardSpeed	41
vr_WideGlanceEyeOffsetX, vr_WideGlanceEyeOffsetY, vr_WideGlanceEyeOffsetZ	41
vr_WideGlanceAdaptiveEyeOffset	41
vr_WideGlanceScreenOffsetX, vr_WideGlanceScreenOffsetY, vr_WideGlanceScreenOffsetZ	41
vr_WideGlanceSensorsBase	41
vr_WideGlanceMonitorWidth, vr_WideGlanceMonitorHeight	41
vr_WideGlanceFOVScale	41
vr_WideGlanceNoFOVChange	41
vr_WideGlanceInvertPerspective	41
vr_WideGlance_StraightMovementScale	42
vr_WideGlance_SideMovementScale	42
WideGlanceStraightMove	42
WideGlanceSideMove	42
Демонстрационные уровни	43
Тестовый кубик	43
Аквариум	44
Гарнизон	45
Камера для стратегий	46
Игра “Space Invaders”	47
Игра “Lesson22” (Тип)	48
Примерный код обработчика ввода	49
Приложение А. Новые возможности устройства - определение направления взгляда (ориентация головы).....	50
Приложение Б. Дополнение в Wideglance SDK после введения расчета вектора направления датчика.....	51
Контакты	52

ВВЕДЕНИЕ В WIDEGLANCE

Торговые марки, патенты

Торговые марки ruCap и WideGlance являются зарегистрированными товарными марками ООО «руКэп» (Россия, Москва).

Технология WideGlance защищена международной патентной заявкой PCT/RU2005/000465 «Способ и система для визуализации виртуальных трехмерных объектов».

Торговые марки “Shine” и “Жор-тур” зарегистрированы компанией ООО “Нью Медиа Дженерейшн” (Россия, Москва).

Глоссарий

ruCap LTD

Российская high-tech компания, созданная в 2005-м году с целью разработки и продвижения на отечественный и мировой рынки новых технологий и устройств, которые обеспечивают объемное восприятие трехмерных виртуальных объектов на экранах настольных или мобильных систем.

Устройство ruCap

Устройство, используемое с обычным монитором и предназначенное для отображения виртуального трехмерного контента под правильным углом зрения. Функционирует благодаря определению абсолютных координат пользователя. Предназначено для индивидуального использования в компьютерных играх и прочих 3D-приложениях.

WIDEGLANCE

Патентованная технология компании ruCap, в основе которой лежит распознавание взаимного положения пользователя и экрана в пространстве.

Антенна

Немобильная часть устройства ruCap – рамка, которая жестко крепится на монитор. Содержит четыре ультразвуковых приемника с платами управления, два инфракрасных излучателя и схемотехнический блок, отвечающий за связь с компьютером при помощи интерфейса USB.

Оголовье

Мобильная часть устройства ruCap. В данном прототипе представляет собой беспроводные наушники, интегрированные со схемотехническим блоком, в котором содержится ультразвуковой передатчик, его плата управления и инфракрасные излучатели.



Физика

Физический движок – часть игрового движка, ответственная за симуляцию физических взаимодействий.

Рендер

Часть игрового движка, ответственная за визуализацию игрового мира.

Стереобаза

Расстояние между камерами для левого и правого глаза при отрисовке изображения в стереорежиме.

Точка наблюдения

По умолчанию – координаты передатчика на оголовье относительно монитора. Если задана модель коррекции положения глаз, то координаты точки наблюдения рассчитываются соответственно заданной модели.

Игровой кадр/фрейм

Игровой кадр/фрейм – один цикл, в течение которого в той или иной последовательности обрабатывается ввод с мыши, клавиатуры и прочих устройств, расчет физических взаимодействий, обработка жизни игровых объектов, сетевая синхронизация и, в итоге, отрисовка 3D-сцены. Некоторые из перечисленных выше процессов могут осуществляться асинхронно по отношению к отрисовке сцены. Нас в данном случае интересует цикл, в который включена отрисовка, т.к. данные, предоставляемые устройством, напрямую влияют на положение камеры и матрицу преобразования.

Камера

1. Относящееся к рендеру понятие, включающее в себя такие параметры, как позиция и ориентация камеры, матрица трансформации, параметры системы отсечения невидимых поверхностей (в частности плоскости отсечения) и т.п.
2. Часть игрового объекта, к которому может привязываться рендер для отрисовки сцены (в режиме от первого или третьего лица). Такими объектами, в частности, являются объект игрока, объект «скриптовая камера». Помимо используемых рендером на текущем кадре параметров типа позиции и ориентации в игровом объекте могут храниться данные, описывающие особенности демпфирования камеры конкретного объекта, связанный с камерой экранный интерфейс и т.п. Параметры WideGlance также могут быть камерозависимыми, например, может отличаться масштаб камер, или для некоторых камер технология WideGlance может быть отключена.

«Режим окна», «режим виртуального окна»

Режим отображения 3D-сцены, при котором смещения головы в реальном мире преобразуются в эквивалентные смещения камеры, а матрица пространственного преобразования строится таким образом, чтобы минимизировать проекционные искажения при текущем положении головы наблюдателя.

Технические детали. Аппаратно-программный комплекс для определения координат пользователя в пространстве ruCar CMU-3

Аппаратная часть

Аппаратная часть комплекса ruCar CMU-3 состоит из двух отдельных устройств. Первое (антенна) крепится на монитор, второе (оголовье) размещается на голове пользователя.

Антенна представляет собой рамку, которая жестко крепится на монитор. Антенну, распространяемую данным документом, предполагается использовать свходящим в комплект SDK 19-дюймовым монитором (в серийном устройстве, благодаря универсальному креплению, пользователь сможет размещать антенну как на 19-ти, так и на 17-ти дюймовых мониторах). Антенна содержит четыре ультразвуковых приемника с платами управления, два инфракрасных излучателя и схемотехнический блок, отвечающий за связь с компьютером при помощи интерфейса USB. Для работы устройства необходимо специальный драйвер, поставляемый в составе комплекта разработчика. Драйвер находится на установочном диске в папке **driver**.

Оголовье в данном прототипе представляет собой беспроводные наушники, интегрированные со схемотехническим блоком, в котором содержится ультразвуковой передатчик, его плата управления и инфракрасные приемники.

Принцип работы

Принцип работы устройства довольно прост:

1. Антенна с помощью инфракрасного (ИК) импульса посылает сигнал Оголовью о начале цикла измерения.
2. Оголовье посылает ультразвуковой (УЗ) сигнал, который принимается каждым из четырех ультразвуковых датчиков Антенны.
3. Процессор определяет время задержки поступления УЗ-импульсов на каждый датчик антенны.
4. Драйвер устройства преобразует время задержки в абсолютные координаты Оголовья относительно антенны.
5. Запускается следующий цикл измерений (система работает с частотой 160 кадров/сек).

- 1 ИК-датчик антенны
- 2 Оголовье
- 3 УЗ-приемники антенны



Программная часть

В программную часть комплекса (драйвер) входит динамически подключаемая библиотека (dll), которая экспортирует функции инициализации, управления и передачи обработанных данных с устройства. Драйвер может подключаться также как статическая библиотека.

Инициализация устройства происходит на этапе загрузки драйвера в приложение. После инициализации можно программным образом изменить установленные по умолчанию параметры устройства: скорость звука, расстояние между датчиками антенны, параметры сглаживания и преобразования потока получаемых данных.

Изменение скорости звука может потребоваться для экстремальных условий эксплуатации устройства – для очень высоких или очень низких температур.

Расстояние между датчиками антенны по вертикали определено конструкцией устройства, а по горизонтали – задается в процессе юстировки устройства непосредственно после его фиксации на мониторе пользователя (в связи с различиями в размерах мониторов у разных производителей).

Изменение параметров сглаживания потока получаемых данных позволяет либо повысить динамические характеристики устройства (корректная обработка быстрых перемещений пользователя), либо повысить его статические характеристики (максимально высокое качество отображения для неподвижного пользователя).

Параметры преобразования задают модель пересчета координат Оголовья в положение точки наблюдения (расположена между глаз пользователя). Данные параметры дополнительно юстируются пользователем для достижения максимальной реалистичности отображения.

Использование

Основная функция устройства состоит в определении абсолютных координат пользователя относительно монитора, что позволяет нам сформировать максимально реалистичное изображение (изображение под правильным углом зрения) трехмерных объектов на экране обычного монитора. Соответственно, у пользователя появляется возможность «заглянуть за угол» в виртуальном мире, а экран монитора становится аналогом прозрачного окна в этот мир.

Возможно использование устройства в качестве манипулятора, формирующего управляющие воздействия в зависимости от перемещения пользователя. Такие перемещения можно, например, интерпретировать как команды для изменения параметров камеры (поворот, зуммирование, смещение) или как команды управления объектами (оружие, спортивные снаряды, прочее).

В некоторых случаях целесообразно совмещение указанных функций. В качестве примера – эффект «заглядывания за угол» можно усилить, если в непосредственной близости от такого угла обрабатывать перемещения пользователя непропорционально.

ВВЕДЕНИЕ В WIDEGLANCE

Соответственно, внедрение технологии WideGlance включает в себя следующие этапы:

1. Определение приемлемых вариантов использования технологии (визуализация, управление объектами, смешанные технологии);
2. Подключение драйвера устройства (динамическая либо статическая библиотека);
3. Внесение изменений в приложение (поддержка нового устройства ввода, модификация игровой логики, физики, соответствующее технологии преобразование 3D-рендера).

Отличия от других устройств ввода. Совместимость

Трекеры. Выдают относительные изменения положения пользователя (не абсолютные). Используются только в качестве манипуляторов. «Заглянуть за угол», используя трекер, не получится, можно только повернуть камеру. Технология WIDEGLANCE позволяет эмулировать режим трекера.

Стереочки. Не являются устройством ввода. Очки используют эффект бинокулярного зрения, им присуща ограниченная «рабочая зона», в которой наблюдается эффект. Используя очки «заглянуть за угол» также не получится. Возможно одновременное использование стереочков и технологии WIDEGLANCE.

Шлемы. Сочетают в себе возможность просмотра стереоизображения и определения поворотов головы. Не определяют смещения головы, обладают ограниченным разрешением изображения, дорого стоят.

ОПИСАНИЕ ФУНКЦИЙ ДРАЙВЕРА RUCAP СМУ-3

Драйвер устройства в настоящее время представляет собой динамически подключаемую библиотеку (dll), которая экспортирует функции инициализации, управления и передачи обработанных данных с устройства.

Драйвер может подключаться также как статическая библиотека.

Перечень экспортируемых функций

Основные документированные функции:

GetSignature
GetDriverVersion
SetSoundSpeed
GetSoundSpeed
SetMask
GetMask
SetFilter
GetFilter
SetMonitorWidth
GetMonitorWidth
GetHeadPos
GetEyeOffset
SetEyeOffset
SetScreenCenterPos
GetScreenCenterPos

Инициализация устройства происходит на этапе загрузки драйвера.

```
wg_result_t GetSignature(LPRUCAPSIGNATUREDATA pSignature)
```

Проверка валидности библиотеки. Функция возвращает структуру, содержащую в себе строку-идентификатор и служащую для однозначного опознания библиотеки.

Параметром функции является указатель на структуру `tagRucapSignature`:

```
typedef struct tagRucapSignature  
{  
    char DllSignature[150];  
} RUCAPSIGNATUREDATA, *LPRUCAPSIGNATUREDATA;
```

```
wg_result_t GetDriverVersion(int& nVersion)
```

Функция возвращает текущую версию драйвера.

ОПИСАНИЕ ФУНКЦИЙ ДРАЙВЕРА RUCAP CMU-3

nVersion – версия драйвера.

Возвращаемое функцией значение всегда WG_RESULT_OK.

wg_result_t SetSoundSpeed(float Speed)

Функция устанавливает значение скорости звука, необходимое для расчета пространственных координат пользователя.

Speed – значение скорости звука в м/с. Возвращаемое функцией значение всегда WG_RESULT_OK.

float GetSoundSpeed()

Функция возвращает установленное в драйвере значение скорости звука в м/с.

wg_result_t SetMask(char Mask)

Для расчета координат пользователя используются данные с четырех ультразвуковых приемников, размещенных на мониторе. Приемники пронумерованы по часовой стрелке, начиная от нижнего левого. Из четырех приемников сформировано четыре комбинации (в каждую комбинацию входят три различных приемника).

Комбинация 1 – 1, 2, 3 датчик.

Комбинация 2 – 2, 3, 4 датчик.

Комбинация 3 – 3, 4, 1 датчик.

Комбинация 4 – 1, 2, 4 датчик.

Функция позволяет исключать из расчетов заданный набор групп.

Mask – битовая маска, задающая набор исключаемых групп. Участвуют 4 первых бита. Включенный бит означает включенную группу.

Возвращаемое функцией значение всегда WG_RESULT_OK.

char GetMask()

Функция возвращает битовую маску групп, используемых для расчета координат пользователя, т.е. значение, установленное функцией SetMask.

wg_result_t SetFilter(int CommonFilter, int SelectedFilter, int ApproxFactor)

Данные, принятые непосредственно с устройства, нуждаются в постобработке. Обработка осуществляется при помощи алгоритмов сглаживания и отсекающего заведомо некорректных значений.

CommonFilter – количество точек для построения аппроксимирующей кривой.

SelectedFilter – количество точек аппроксимирующей кривой, из которых будут удалены точки с минимальными и максимальными значениями и усреднены оставшиеся.

ApproxFactor – данное значение должно быть меньше или равно SelectedFilter. Значение показывает, какое число точек из SelectedFilter будет усреднено.

Заданные значения по умолчанию (20;10;6).

Возвращаемое функцией значение всегда WG_RESULT_OK.

`wg_result_t GetFilter(int&CommonFilter,int&SelectedFilter,int&ApproxFactor)`

Функция возвращает параметры для постобработки полученных с устройства данных.

Возвращаемое функцией значение всегда WG_RESULT_OK.

`wg_result_t SetMonitorWidth(float W)`

Функция устанавливает расстояние между двумя симметрично расположенными по горизонтали приемниками на мониторе.

W – расстояние между датчиками в метрах.

`float GetMonitorWidth()`

Функция возвращает расстояние (в метрах) между двумя симметрично расположенными по горизонтали приемниками на мониторе.

`int GetHeadPos(float& x,float&y,float& z)`

Функция возвращает координаты пользователя относительно монитора. Центр начала координат находится между двумя средними датчиками на мониторе.

x – координата по оси X.

y – координата по оси Y.

z – координата по оси Z.

Возвращаемые функцией значения:

WG_RESULT_OK – значение возвращается при успешном получении и обработке координат.

WG_RESULT_FAIL – значение, возвращаемое в случае ошибки.

`wg_result_t SetEyeOffset(int nModelCode,float x,float y,float z)`

Функция устанавливает модель коррекции положения глаз пользователя относительно приемника и требуемые для нее параметры.

Переменная nModelCode может принимать значения:

WG_MODEL_NO – корректировка положения глаз не применяется.

WG_MODEL_ORTO – ортогональная корректировка.

WG_MODEL_SPHERICAL – сферическая корректировка.

WG_MODEL_ANATOMICAL – анатомическая корректировка.

Детальное описание моделей корректировки приведено далее по тексту.

x, y, z – параметры модели.

`wg_result_t GetEyeOffset(int&nModelCode,float&x,float&y,float&z)`

Функция возвращает текущую модель корректировки положения глаз с установленными для нее параметрами.

Переменная nModelCode может принимать значения:

WG_MODEL_NO – корректировка положения глаз не применяется.

ОПИСАНИЕ ФУНКЦИЙ ДРАЙВЕРА RUCAP СМУ-3

WG_MODEL_ORTO – ортогональная корректировка.

WG_MODEL_SPHERICAL – сферическая корректировка.

WG_MODEL_ANATOMICAL – анатомическая корректировка.

Детальное описание моделей корректировки приведено далее по тексту.

x, y, z – параметры модели.

wg_result_t SetScreenCenterPos(float x,float y,float z)

Функция устанавливает смещение центра экрана относительно точки располагающейся посередине между датчиками. Это смещение определяется типом монитора (конструкцией рамки монитора). X – экранная горизонталь, Y – вертикаль, Z – нормаль к экрану.

wg_result_t GetScreenCenterPos(int&nModelCode,float&Sx,float&Sy,float&Sz)

Функция возвращает смещение центра экрана относительно точки, располагающейся посередине между датчиками.

Программные компоненты SDK

ruCap.h

Путь к файлу “Include\Rucap.h”

Описание прототипов функций драйвера, типов данных, констант.

ruCap.dll

Путь к файлу “dll\Rucap.dll”

Драйвер в виде динамически подключаемой библиотеки.

ruCap.Lib

Путь к файлу “Lib\Rucap.lib”

Драйвер в виде статической библиотеки.

rucap.inf, rucap.sys

Путь к файлам “driver\rucap.inf, driver\rucap.sys”

Системный драйвер устройства. Устанавливается при первом подключении устройства.

ИСПОЛЬЗОВАНИЕ ВЫДАВАЕМЫХ ДРАЙВЕРОМ ДАННЫХ. ПЕРВИЧНАЯ ОБРАБОТКА ДАННЫХ

Драйвер устройства в настоящее время представляет собой динамически подключаемую библиотеку (dll), отвечающую за управление устройством, сбор и первичную обработку данных. Возможно также оформление драйвера в виде статической библиотеки.

Загрузка драйвера

Загрузка драйвера производится обычно при запуске приложения одновременно с инициализацией других устройств ввода (клавиатура, мышь, джойстик).

Прототипы функций драйвера описаны в прилагаемом файле [ruCap.h](#)

Пример использования драйвера – см. прилагаемые файлы

[CMyWideGlanceDriver.h](#), [CMyWideGlanceDriver.cpp](#)

```
// Макрос запроса функции по имени из DLL
#define LD(name) fnWideGlance##name=reinterpret_
cast<tfnWideGlance##name*>(GetProcAddress(m_dll,#name))
// Макрос проверки корректности указателя на функцию
#define null_fn(name) ((fnWideGlance##name)==NULL)
```

Сами указатели на функции драйвера лучше разместить в теле объекта, занимающегося работой с драйвером (или сделать static-членами соответствующего класса). Например, так:

```
class CWin32WideGlanceDriver : public CWideGlanceDriver
{
protected:
    HINSTANCE m_dll; // DLL драйвера
public:
    wg_def_fn(GetSignature);
    wg_def_fn(GetDriverVersion);
    [пропущено]
};
```

После того, как библиотека драйвера загружена нужно произвести проверку строки-сигнатуры (функция `GetSignature`), версии драйвера (функция `GetDriverVersion`), установить указатели на остальные функции и вызвать функцию `SetMonitorWidth`.

ИСПОЛЬЗОВАНИЕ ВЫДАВАЕМЫХ ДРАЙВЕРОМ ДАННЫХ

```

// примерный код - для указания на место, в которое должен
// вставляться вызов загрузки драйвера.
void CMyApp::mfOpenInputDevice()
{
    mcpMouse = NULL;
    mcpJoystick = NULL;
    if (!mcpMouse)
    {
        mcpMouse = new CMyMouseDriver();
        if (mcpMouse)
            mcpMouse->mfOpen();
    }

    mcpJoystick = new CMyJoystickDriver();
    mcpJoystick->mfOpen();

    [прочие устройства]
    // создание объекта для работы с драйвером
    mcpWideGlanceDriver = new CMyWideGlanceDriver();
    //загрузка и инициализация драйвера
    mcpWideGlanceDriver->mfOpen();
}

// процедура открытия устройства и инициализации драйвера
bool CMyWideGlanceDriver::mfOpen ()
{
    if (mbOpened)
        return true;
    if (!vr_WideGlanceEnable)
        return false;
    if (!m_dll)
        mfLoadDLL();
    if (!m_dll)
        return false;
    mbOpened = true;
    if (fnWideGlanceSetMonitorWidth)
    {
        float fWidth = CV_vr_WideGlanceSensorsBase.mfGetVal();
        wg_result_t result = fnWideGlanceSetMonitorWidth(fWidth);
        if (result != WG_RESULT_OK)
            return false;
    }
    return true;
}

```

```

// загрузка библиотеки драйвера
void CMyWideGlanceDriver::mfLoadDLL()
{
    const char* dllname = "rucap"
    m_dll = LoadLibrary(dllname);
    if (m_dll == NULL)
        return;
    // Получение указателей на функции
    LD(GetSignature);
    if(!null_fn(GetSignature))
    {
        static const char szSignature[] =
            "ruCapGDT-1 DeviceDriver\nCopyright ruCap\nRUCAP.DLL\n
            nCopyright ruCap";
        // проверяем строку-сигнатуру
        RUCAPSIGNATUREDATA pSignature;
        wg_result_t result=fnWideGlanceGetSignature(&pSignature);
        if( result == WG_RESULT_OK )
        {
            if(strncmp(szSignature,pSignatureDISignature,SIGNATURE_
                LEN)==0)
            {
                result = WG_RESULT_OK;
                // проверяем версию драйвера
                LD(GetDriverVersion);
                if(!null_fn(GetDriverVersion))
                {
                    int version;
                    result=fnWideGlanceGetDriverVersion(version);
                    if(version == RUCAP_VERSION)
                    {
                        // связываем остальные функции
                        LD(SetSoundSpeed);          LD(SetMask);
                        LD(SetFilter);              LD(GetFilter);
                        LD(SetMonitorWidth);        LD(GetHeadPos);
                        LD(GetEyeOffset);           LD(SetEyeOffset);
                        LD(SetScreenCenterPos);     LD(GetScreenCenterPos);
                    }
                    if(
                        null_fn(SetSoundSpeed) || null_fn(SetMask) ||
                        null_fn(SetFilter) || null_fn(GetFilter) ||
                        null_fn(SetMonitorWidth)||null_fn(GetHeadPos)||
                        null_fn(GetEyeOffset) || null_fn(SetEyeOffset)||
                        null_fn(GetScreenCenterPos)||
                        null_fn(SetScreenCenterPos))
                    {
                        FreeLibrary(m_dll);
                        m_dll = NULL;
                    }
                }
            }
        }
    }
}

```


Использование сферической системы координат

Предполагается, что взгляд направлен всегда в центр экрана, а линия глаз всегда горизонтальна. Для внесения поправки нужно определить систему координат, связанную с головой, – векторы вперед, вверх, вправо. Вектор вперед направлен из глаза в центр экрана, вектор вправо – горизонтален.



Анатомические модели

Сидящий человек

При перемещениях в пределах 30 см к экрану, 20 см от экрана и 20 см в стороны можно с хорошим приближением считать, что взгляд направлен в центр экрана. Перемещения головы можно описать как перемещения точки на свободном конце шарнирно-закрепленного стержня. При больших смещениях к экрану к перемещениям головы подключается шея (наклон головы вперед). При больших смещениях от экрана происходит вытягивание шеи. При больших смещениях вбок – перемещение дополняется боковыми наклонами головы. Положение человека перед экраном – не обязательно строго по центру. Смещения рабочего места до 15 см в сторону возможны и не воспринимаются как сколько-нибудь неудобное положение. Допустимые смещения рабочего места вперед-назад составляют примерно 10-15 см. Анатомическая модель позволяет получить более точное положение глаз, но для этого нужно знать местоположение человека.

Стоящий человек

Спортивные, танцевальные симуляторы могут предполагать, что человек не сидит перед монитором, а стоит. В этом случае потребуется другая анатомическая модель, разработку которой мы оставляем на ваше усмотрение.

Усовершенствования анатомической модели

При положениях головы близких к экранной плоскости, большой вклад в восприятие картинки начинает вносить глаз, который находится дальше от экранной плоскости. В этом случае предположение о расположенной на переносице «точке наблюдения», для которой генерируется 3D-сцена, оказывается неверным – положение «точки наблюдения» смещается в направлении от экранной плоскости. Естественно, этого не требуется при стереорежиме – в этом случае картинка рисуется для каждого глаза.

Стереоизображение

Вычисление координат глаз для стереоизображений не представляет труда – все модели подразумевают определение полной ориентации головы, то есть связанного с головой репера (вверх-вниз, вперед-назад, влево-вправо). Для получения координат глаз необходимо отложить от положения «среднего глаза» вправо и влево по половине величины стереобазы.

Обработка данных

Результат, возвращенный функцией `GetHeadPos`, нуждается в дальнейшей обработке. Требуется перевести координаты головы в используемую в клиентском 3D-приложении систему координат, скорректировать погрешность, возникающую из-за несовпадения положений излучателя и глаз, а также, при необходимости, скорректировать ширину поля зрения.

Вы можете вычислять положение глаз, используя модель, приспособленную под нужды вашего приложения, или воспользоваться предоставляемыми драйвером способами коррекции.

Для этого перед функцией `GetHeadPos` надо вызывать функцию `SetEyeOffset`, в параметрах которой указать используемую модель коррекции положения глаз и смещения глаза относительно излучателя.

В функцию первичной обработки данных полезно ввести код, отвечающий за настройки ширины поля зрения, изменения ширины поля зрения, перспективы. Это позволит геймдизайнеру выбрать наиболее удобный способ отображения. Предполагается, что задачей игрового программиста в данном случае является предоставление гейм-дизайнеру максимальных возможностей для осуществления настроек и экспериментов с целью выбора оптимально подходящий для конкретной задачи режим.

```
// Опрос устройства. Метод вызывается на каждом кадре
void CMyWideGlanceDriver::mfTick()
{
    if (!mbOpened)
        return;
    if (!vr_WideGlanceEnable)
        return;
    float fHeadX, fHeadY, fHeadZ;
    if (fnWideGlanceSetEyeOffset)
    {
        wg_result_t result =
            fnWideGlanceSetEyeOffset(WG_MODEL_SPHERICAL,
                vr_WideGlanceEyeOffsetX,
                vr_WideGlanceEyeOffsetY,
                vr_WideGlanceEyeOffsetZ);
    }
    if (fnWideGlanceScreenCenterPos)
    {
```

```

        wg_result_t result =
        fnWideGlanceSetScreenCenterPos(vr_WideGlanceScreenOffsetX,
                                       vr_WideGlanceScreenOffsetY,
                                       vr_WideGlanceScreenOffsetZ);
    }
    if(fnWideGlanceGetHeadPos)
    {
        wg_result_t result=fnWideGlanceGetHeadPos(fHeadX,fHeadY,
fHeadZ);
        vr_WideGlance_OriginalXPos = fHeadX;
        vr_WideGlance_OriginalYPos = fHeadY;
        vr_WideGlance_OriginalZPos = fHeadZ;
    } else
        return;
//определяем направления экранных осей в мировых координатах
// vecScreenX - экранная горизонталь
// vecScreenY - экранная вертикаль
// vecScreenN - нормаль к экрану
    CVec3 vecScreenX(1, 0, 0);
    float fScreenW = fWidth;
    CVec3 vecScreenY(0, 1, 0);
    vecScreenY.Normalize();
    CVec3 vecScreenN(0, 0, -1);
    vecScreenN.Normalize();
// находим центр экрана
    CVec3 vecScreenCenter(0, 0, 0);
//точка перед экраном, привзгляд из которой видимая ширина экрана
// будет равна той, что установлена в игре по умолчанию
    CVec3 vecZeroPos;
// ограничиваем поле зрения разумными пределами
    float fFOV = gf_FOV;
    if(fFOV < 1)
        fFOV = 1.0;
    else
        if(fFOV > 160)
            fFOV = 160;
    float fDist=0.5*vr_WideGlanceMonitorWidth/tan(fFOV/360*M_PI);
    vecZeroPos = vecScreenCenter - fDist * vecScreenN;
    float fFOVScale = vr_WideGlanceFOVScale;
    CVec3 vecHeadPos(fHeadX, fHeadY, fHeadZ);
// придвигаем голову к экрану по нормали
    vecHeadPos.z /= fFOVScale;
//определяем масштаб расстояния от калибровочного положения головы
// до центра экрана - 1)
    CVec3 vecZeroDir = vecScreenCenter - vecZeroPos;

```

■ ПЕРВИЧНАЯ ОБРАБОТКА ДАННЫХ

```

float fScale = vecZeroDir.Size();
CVec3 vecHeadOffset;
if(fScale > 0.0001)
    vecHeadOffset=(vecHeadPos-vecZeroPos)*(1.0/fScale);
else
    vecHeadOffset = CVec3(0,0,0);
// vecHP - координаты головы в системе координат, связанной
// с нормальным положением камеры. В случае, когда не используется
// технология WideGlance камера в этой системе имеет координату (0,0,0),
// точке в центре экрана соответствует координата (0, 0, 1)
CVec3 vecHP;
vecHP.x = DotProduct(vecHeadOffset, vecScreenX);
vecHP.y = DotProduct(vecHeadOffset, vecScreenY);
vecHP.z = DotProduct(vecHeadOffset, vecScreenN);
// если включено игнорирование изменения ширины поля зрения
if(vr_WideGlanceNoFOVChange != 0)
    vecHP.z = 0;
if(vr_WideGlanceInvertPerspective != 0)
    vecHP.z = -vecHP.z;
vr_WideGlance_XPos = vecHP.x;
vr_WideGlance_YPos = vecHP.y;
vr_WideGlance_ZPos = vecHP.z;
CVec3 vecOffset = vecHP - m_vecPrevHeadPos;
m_vecPrevHeadPos = vecHP;
mfDo_SendFloat(vecOffset.z, 0);
mfDo_SendFloat(vecOffset.x, 1);
}

```

ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ WIDEGLANCE В ИГРОВОМ ДВИЖКЕ (НА ПРИМЕРЕ SHINE)

Действия во время серверного фрейма

Физика

Обработка физических взаимодействий (физика) в данном случае отделена от обработки логики игровых объектов. В ходе проведенных исследований выяснилось, что для качественного использования технологии в физику требуется внести небольшие изменения.

Зачем потребовались изменения в физике

В процессе внедрения технологии в игровой продукт выяснилось, что при смещении точки наблюдения и, соответственно, при смещении связанной с объектом игрока камеры, возникает необходимость в перемещении коллизионного объема игрока (появляются ошибки при уклонении от снарядов противника; при смещении головы в сторону не получается пройти в узкий дверной проем; можно свалиться в пропасть из-за неверного определения точки опоры). Также мы пришли к выводу, что при смещении камеры требуется сместить и точку вылета снарядов, из чего следует, что при проверке положения камеры нужно ориентироваться уже не на величину ZNear, а на максимальный коллизионный объем выстреливаемого снаряда, иначе при прижимании камеры к стене ракеты будут взрываться у нас в руках.

В связи с этим было принято решение организовать синхронное перемещение коллизионного объема игрока.

Возникшие сложности

Перемещение не должно быть инерционными, при прекращении движения головы движение объекта игрока должно тут же прекратиться. Смещение головы не должно приводить к изменению скорости игрока, иначе при резких движениях головы возможно развить такую скорость у персонажа, что при ударе о стену будет наноситься повреждение. Попытка реализовать смещение через изменение скорости выявила еще один неприятный эффект – влияние трения и дрейф коллизионного объема. Небольшие смещения головы и шумы (неравномерность фреймрейта, аппаратные шумы) вызывали хаотическое изменение скорости, хотя и небольшое по величине, но не всегда одинаково компенсировавшееся трением (неравномерности фреймрейта в том числе были причиной). В итоге игрок начинал медленное «броуновское движение» по полу, что было неприемлемо – появлялась за-

держка на кадр. Обработка смещения головы могла быть проведена только в тот момент, когда управление передается объекту игрока, после симуляции физических взаимодействий. Изменение скорости сказывалось только на следующем фрейме и, ввиду неравномерности фреймрейта, смещение игрока оказывалось довольно приблизительным. По направлению оно, конечно, совпадало, но по величине могло отличаться в разы. Его приходилось усреднять, но хорошо все равно не получалось.

Было принято решение – в дополнение к инерционному перемещению игрока (обусловленному текущей скоростью) сделать неинерционное. В физику добавить величину неинерционного смещения за кадр. Каждый кадр это смещение используется, обнуляется и при обработке ввода заполняется снова. Т.к. смещение исчисляется в абсолютных величинах и действует только один кадр, то задержка на кадр и неравномерности фреймрейта влияния не оказывают.

Тонкости внедрения неинерционного смещения

Стазис. Объекты с нулевой скоростью переводятся в особое состояние, именуемое стазисом. В этом состоянии они не просчитывают коллизии с окружающими предметами и выходят из этого состояния либо при столкновении с каким-либо объектом, либо при получении импульса, либо прямым вызовом снятия стазиса. Неинерционное перемещение должно вызывать выход из стазиса.

Скольжение вдоль стен. При столкновении со стеной или другим препятствием происходит изменение вектора скорости. Если после столкновения скорость нулевая, то обработка физики прекращалась. При наличии неинерционного перемещения это место требует доработки.

Изменения в физике

- Добавление переменной – неинерционного смещения на текущем кадре – 1 строка
- Инициализация этой переменной в конструкторе – 1 строка
- Сбрасывание флага стазиса при наличии неинерционного смещения – 1 строка
- Определение смещения объекта за кадр. К произведению средней скорости на время добавляется неинерционное смещение – несколько строк по числу способов перемещения
- Обработка столкновения – остановка происходит при условии нулевого вектора скорости и нулевого вектора неинерционного смещения – 1 строка
- скольжение вдоль стены – производится клиппинг вектора неинерционного смещения – 1 строка

Игровая логика

Оружие

Доработка оружия включает в себя:

Перемещение модели оружия

При смещении головы мы не должны видеть оружие с изнанки (там может просто не быть полигонов). Камера не должна далеко отодвигаться от оружия (будут видны обрезанные руки). Камера не должна приближаться слишком близко к оружию (не ближе Znear). В общем случае необходим постоянный контроль положения модели оружия относительно камеры.

Перемещение точки вылета снарядов

Для того чтобы снаряды вылетали всегда из ствола, в функцию, определяющую точку вылета, надо добавить поправку, учитывающую смещение головы. Если точка вылета снарядов определяется координатами специальной кости на модели оружия, то этот пункт решается автоматически.

Изменение направления вылета снарядов

Неоднозначный момент – должно ли поворачиваться оружие при смещении головы. Если оружие у нас в руках, то, скорее всего, да. Но если рассматривать что-то наподобие танкового симулятора, то может быть и нет. Соответственным образом надо поворачивать модель оружия.

Перемещение или неперемещение прицела

Должно быть синхронизировано с изменением направления вылета снарядов.

Снайперский прицел

В снайперском режиме может потребоваться отключение использования технологии.

Используемые объекты

В данном случае рассматриваются объекты, лежащие в инвентаре и получающие управление при нажатии кнопки «использовать». При этом исполняется специфический для конкретного объекта код.

Например, фонарик. Направление свечения фонарика может определяться направлением взгляда объекта игрока (фонарик на голове – при смещении головы фонарик поворачиваем). Или направлением туловища объекта игрока (при смещении головы направление луча фонарика не изменяется или в меньшей степени относительно смещения головы). В целом изменения аналогичны изменениям в оружии.

В «Жор-туре» не было других подобных объектов. Но дело в том, что тут управление передается объекту, и описать таких объектов можно сколь угодно много (например, в RPG). Для использования в подобных местах из рендера экспортируется специальная функция, возвращающая направление в центр экрана. Вместо нормали к экрану нужно (или не нужно – по обстоятельствам) будет использовать результат этой функции.

Обработка ввода

Команды WideGlanceStraightMove, WideGlanceSideMove обрабатываются объектом игрока, и результатом этой обработки становится величина неинерционного смещения в физике. Этот процесс происходит во время вызова gcCmd.mfExecute () до серверного фрейма и, соответственно, до работы физики. Поэтому не появляется задержка на кадр.

Перемещение

Вопрос, ответ на который отнюдь неоднозначен, – куда должен перемещаться игрок при нажатии клавиши «вперед» в случае если он сместил голову в сторону. В центр экрана или в направлении нормали к экрану? Большинству опрошенных нами игроков (не менее 70%) понравился вариант с перемещением в направлении центра экрана. Но были и те, кому понравился другой вариант. Эти рассуждения верны только для вида от первого лица. При включенном виде от третьего лица изменение положения головы не должно влиять на направление перемещения объекта игрока.

Взаимодействие с объектами среды. Подбирание предметов

При взаимодействии с объектами используется предмет видимый в центре экрана.

За эту функциональность отвечает отдельный код, обрабатывающий в объекте игрока команду «использовать». Этот код ищет используемый объект перед игроком. Для поиска ранее использовалось направление нормали к экрану, а теперь – направление в центр экрана.

Наведение на объекты

В случае подсветки предмета, на который наводится игрок, подсветка должна появляться у объекта, находящегося в центре экрана. Это не тот же самый программный код, что в предыдущем пункте. Производится поиск объекта лежащего перед игроком (по тому же самому алгоритму), но объект проверяется не на возможность использования, а на наличие в нем сообщения, выводимого на экран над объектом.

Прочие объекты

Это, прежде всего, объекты, в которые может помещаться камера. Для таких объектов может понадобиться отключение технологии. Например, игровая камера наблюдения показывается как изображение на экране монитора, соответственно, никакого «эффекта окна» быть не должно. Скриптовые камеры могут как включать технологию, так и выключать. Отключение технологии позволят использовать такой мощный изобразительный прием, как изменение ширины поля зрения. При отключенной технологии во время просмотра скриптового ролика игрок гарантированно увидит то, что хотели показать ему разработчики.

Действия при подготовке к рендеру и во время рендера

В объект рендера был включен вектор, описывающий смещение камеры относительно нормального положения. При подготовке к рендеру сцены в этот вектор собираются значения из консольных переменных `vr_WideGlance_XPos`, `vr_WideGlance_YPos`, `vr_WideGlance_Zpos`. Хранение координат камеры в отдельных консольных переменных было сделано для облегчения работы с ними из внешнего скрипта.

```
if(vr_WideGlanceEnable.mfGetVal())
{
    gsRefView.m_vecWideGlancePosition.x=vr_WideGlance_XPos;
    gsRefView.m_vecWideGlancePosition.y=vr_WideGlance_YPos;
    gsRefView.m_vecWideGlancePosition.z=vr_WideGlance_ZPos;
} else {
    gsRefView.m_vecWideGlancePosition.x = 0;
    gsRefView.m_vecWideGlancePosition.y = 0;
    gsRefView.m_vecWideGlancePosition.z = 0;
}
```

После этого управление передается объекту рендера.

Практически, первое, что делает рендер, это вызов функции `void CRenderDllGlobals::sfCalcView(void)`, которая отвечает за вычисление параметров пирамиды видимости.

```
void CRenderDllGlobals::sfCalcView(void)
{
    float x;
    float fov_x, fov_y;
    fov_x = nCV_v_FOV;
    if ( fov_x < 1 )
        fov_x = 1;
    else
        if ( fov_x > 160 )
            fov_x = 160;
    x = gsRD.vrect.width / tan( fov_x / 360 * M_PI );
    fov_y = atan2( gsRD.vrect.height, x );
    fov_y = fov_y * 360 / S_PI;
    // set it
    gsRD.horizontalFOV = fov_x;
    gsRD.verticalFOV = fov_y;
    // получаем текущий контроллер камеры
    CECameraController* pCameraController =
        CECameraController::sfGetCameraController();
    if (pCameraController)
```

ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ WIDEGLANCE В ИГРОВОМ ДВИЖКЕ

```

{
    CVec3 vecViewPosition;
    CQuaternion qViewRotation;
    pCameraController->mfCalcViewPosition(
        &vecViewPosition, &qViewRotation);
    if(pCameraController->m_bUseWideGlance)
    {
        // WideGlance
        CVec3 vecForward, vecRight, vecUp;
        qViewRotation.ToPivot( &vecForward, &vecRight, &vecUp );
        CVec3 vecWideGlancePosition=gsRD.m_vecWideGlancePosition;
        CVec3 vecWideGlanceCameraOffset =
            gsRD.m_vecWideGlancePosition *
            pCameraController->m_fWideGlance_CameraScale;
        CVec3 vecOriginalPos = vecViewPosition;
        // корректируем положение камеры в мировых координатах
        CVec3 vecOffset=vecForward*vecWideGlanceCameraOffset.z-
            vecRight*vecWideGlanceCameraOffset.x +
            vecUp*vecWideGlanceCameraOffset.y;

        vecOffset.x*pCameraController->m_vecWideGlance_AxesMovementx;
        vecOffset.y*pCameraController->m_vecWideGlance_AxesMovementy;
        vecOffset.z*pCameraController->m_vecWideGlance_AxesMovementz;
        vecViewPosition += vecOffset;
        // проверяем - можно ли сместить камеру в сторону
        // (проверка на залезание в стену)
        vecViewPosition =
            pCameraController->mfTraceCamera(vecOriginalPos,
            vecViewPosition);
        // корректируем FOV
        CVec3 vecToScreen=CVec3(0,0,1)-vecWideGlancePosition;
        float fKatet=tan(DEGTORAD(fov_x)/2.0)/vecToScreen.Size();
        CVec3 vecNewForward, vecNewRight;
        CVec3 vecNewUp, vecNewLeft, vecNewDown;
        vecNewForward=vecForward*(1-vecWideGlancePosition.z);
        vecNewRight = vecRight * (1 - vecWideGlancePosition.x);
        vecNewLeft = vecRight * (-1 - vecWideGlancePosition.x);
        vecNewUp = vecUp * (1 - vecWideGlancePosition.y);
        vecNewDown = vecUp * (-1 - vecWideGlancePosition.y);
        float fKatetX = tan(DEGTORAD(fov_x)/2.0);
        float fKatetY=fKatetX*gsRD.vrect.height/gsRD.vrect.width;
        gsRD.m_fTopFOVBound =
            (atan((fKatetY - vecWideGlancePosition.y)/
            (1 - vecWideGlancePosition.z))) * 180 / S_PI;
    }
}

```

```

gsRD.m_fBottomFOVBound =
    (atan((-fKatetY - vecWideGlancePosition.y)/
    (1 - vecWideGlancePosition.z))) * 180 / S_PI;
gsRD.m_fRightFOVBound =
    (atan((fKatetX - vecWideGlancePosition.x)/
    (1 - vecWideGlancePosition.z))) * 180 / S_PI;
gsRD.m_fLeftFOVBound =
    (atan((-fKatetX - vecWideGlancePosition.x)/
    (1 - vecWideGlancePosition.z))) * 180 / S_PI;
gsRD.horizontalFOV = fov_x;
gsRD.verticalFOV = fov_y;
// WideGlance
} else {
    if(nCV_r_CorrectedVisMatrix)
    {
        gsRD.m_fTopFOVBound = fov_y * 0.5;
        gsRD.m_fBottomFOVBound = -fov_y * 0.5;
        gsRD.m_fRightFOVBound = fov_x * 0.5;
        gsRD.m_fLeftFOVBound = -fov_x * 0.5;
    } else {
    }
}
gsRD.vieworg = vecViewPosition;
gsRD.viewangles = qViewRotation.ToPitchYawRollDegrees();
}
}

```

После этой функции в рендере обновляются значения переменных

```

gsRD.m_fTopFOVBound
gsRD.m_fBottomFOVBound
gsRD.m_fRightFOVBound
gsRD.m_fLeftFOVBound

```

Эти переменные соответствуют углам раскрытия сторон пирамиды видимости относительно направления взгляда камеры.

Следующее место, потребовавшее внесения изменений – построение матрицы преобразования.

```

void CRender3DD3D:mfd3DSetSceneProjMatrix(float zNear, float zFar)
{
    float xMin, xMax, yMin, yMax, zMin, zMax;
    float xDist, yDist, zDist;
    zMax = g_Zfar;
    zMin = g_Znear;

```

ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ WIDEGLANCE В ИГРОВЫМ ДВИЖКЕ

```

const float      c_fSin37_5 = 0.6087614f;
yMax = zMin * tan( gsRD.m_fTopFOVBound * S_PI / 180.0 );
yMin = zMin * tan( gsRD.m_fBottomFOVBound * S_PI / 180.0 );
xMax = zMin * tan( gsRD.m_fRightFOVBound * S_PI / 180.0 );
xMin = zMin * tan( gsRD.m_fLeftFOVBound * S_PI / 180.0 );;
xDist = xMax - xMin;
yDist = yMax - yMin;
zDist = zMax - zMin;

```

[пропущено несущественное]

```

D3DXMatrixPerspectiveOffCenterRH(
    (D3DXMATRIX *)(&gsCI.ProjMatrix[0][0]),
    xMin, xMax,
    yMin, yMax,
    zMin, zMax);

```

[пропущено несущественное]

```

m_pD3DDevice->SetTransform(D3DTS_PROJECTION(D3DMATRIX*)(&gsCI.
ProjMatrix[0][0]) );
}

```

После внесения этих изменений изображение на экране уже будет соответствовать «режиму окна».

Для игровых объектов из рендера экспортируется функция

```

void CRenderDIGlobal::CorrectView(CECameraController* pCameraController,
    const CVec3& vecOriginIn, const CQuaternion& qRotationIn,
    CVec3& vecOriginOut, CQuaternion& qRotationOut,
    float fFovXIn,
    float& fFovXOut, float& fFovYOut)
{
    if(pCameraController->m_bUseWideGlance)
    {
        CVec3 vecForward, vecRight, vecUp;
        qRotationIn.ToPivot( &vecForward, &vecRight, &vecUp );
        CVec3 vecWideGlancePosition=gsRD.m_vecWideGlancePosition;
        CVec3 vecWideGlanceCameraOffset=gsRD.m_vecWideGlancePosition*
            pCameraController->m_fWideGlance_CameraScale;
        // корректируем FOV
        CVec3 vecToScreen = CVec3(0,0,1) - vecWideGlancePosition;
        float fKatet = tan(DEGTORAD(fFovXIn)/2.0) / vecToScreen.Size();
        CVec3 vecNewForward, vecNewRight, vecNewUp;
        vecNewForward = vecForward;
    }
}

```

```

vecNewForward -= vecForward * vecWideGlancePosition.z;
vecNewForward += vecRight * vecWideGlancePosition.x;
vecNewForward -= vecUp * vecWideGlancePosition.y;
vecNewForward.Normalize();
CrossProduct(vecUp, vecNewForward, vecNewRight);
vecNewRight.Normalize();
CrossProduct(vecNewForward, vecNewRight, vecNewUp);
vecNewUp.Normalize();
qRotationOut.FromPivot(vecNewForward,vecNewRight,vecNewUp);
float fKatetX = fKatet * DotProduct(vecNewRight, vecRight);
fFovXOut = atan(fKatetX) * 360 / S_PI;
if(fFovXOut > pCameraController->m_fWideGlance_MaxFOV)
    fFovXOut = pCameraController->m_fWideGlance_MaxFOV;
floatfKatetY=fKatet*DotProduct(vecNewUp,vecUp)*gsRD.vrect.height/
    gsRD.vrect.width;
fFovYOut = atan(fKatetY) * 360 / S_PI;
if(fFovYOut > pCameraController->m_fWideGlance_MaxFOV)
fFovYOut = pCameraController->m_fWideGlance_MaxFOV;
// корректируем положение камеры в мировых координатах
CVec3vecOffset=vecForward*vecWideGlanceCameraOffset.z-
    vecRight*vecWideGlanceCameraOffset.x +
    vecUp*vecWideGlanceCameraOffset.y;
vecOffset.x*=pCameraController->m_vecWideGlance_AcesMovementx;
vecOffset.y*=pCameraController->m_vecWideGlance_AcesMovementy;
vecOffset.z*=pCameraController->m_vecWideGlance_AcesMovementz;
vecOriginOut = vecOriginIn + vecOffset;
// проверка на залезание в стену
vecOriginOut=pCameraController->mfTraceCamera(vecOriginIn,
vecOriginOut);
} else {
    vecOriginOut = vecOriginIn;
    qRotationOut = qRotationIn;
    fFovXOut = fFovXIn;
    fFovYOut = fFovXIn * gsRD.vrect.height / gsRD.vrect.width;
}
}

```

Перемещение коллизионного объема (физического тела) синхронно с головой

Первый подход – применение силы

На каждом игровом фрейме к телу игрока прикладывается некоторая сила, направленная соответственно смещению головы. Этот подход позволяет избежать больших изменений в коде, отвечающем за физику, т.к. приложение силы к объекту – обычное дело в физических движках. Но встает вопрос о величине этой силы. Величину можно посчитать, зная длительность фрейма и массу. Получится, что в течение фрейма тело игрока разогналось до такой скорости, что смещение за фрейм составило именно такую величину. Для корректной работы алгоритма требуется хранить скорость смещения головы на предыдущем фрейме и вычислять ускорение.

Минусы подхода

- Зависимость от фреймрейта. Требуется проверять алгоритм на корректность работы при различных значениях fps
- Зависимость от неравномерности fps
- Задержка на кадр. Физика отрабатывает силу, вычисленную по результатам предыдущего кадра
- Добавление новой силы влияет на скорость игрока. В играх максимальная скорость игрока часто ограничивается
- Возможность «убить себя головой о стену». При резких движениях головой можно столкнуться с виртуальным препятствием на очень большой скорости
- На итоговое смещение оказывает влияние трение

Плюсы

- Возможно, изменения в физике не потребуются.

Второй подход – неинерционное движение

К параметрам физики (скорость, ускорение, масса...) добавляется вектор неинерционного смещения за кадр. Это смещение добавляется в параметры физики игрока при обработке ввода. В конце физического фрейма смещение обнуляется. В тот момент, когда производится вычисление смещения объекта, к этому смещению добавляется неинерционный сдвиг. То есть формально скорость объекта остается той же, но оказывается он стремится дальше (или ближе). Важная деталь: если в физике есть понятие стазиса, то при обработке ввода признак состояния стазиса у объекта игрока должен быть сброшен, а при проверке на стазис должен учитываться и вектор неинерционного смещения.

Минусы подхода

- Требует модификации физического движка

Плюсы

- Теоретически результат должен слабо зависеть от fps
- Нет задержки на кадр
- Нет влияния на скорость. Следовательно, на величину смещения не оказывает влияния трение

Как это выглядит в движке:

Ввод данных (опрос устройств) – определяется, какие кнопки были нажаты, на сколько переместилась мышь, куда и на сколько сместилась голова.

Преобразование введенных данных в консольные команды. Производится на основе текущих настроек управления – по нажатию такой-то кнопки производится то-то и то-то, при смещении головы вперед производится смещение объекта игрока вперед.

Интерпретация консольных команд. При обработке консольной команды на перемещение объекта игрока вперед. В объект игрока записывается соответствующее текущему кадру смещение:

```

if(strcmp(szKey, "StraightMove")==0)
{
    float x;
    int nscan = sscanf(szValue, "%f", &x);
    if(nscan == 1)
    {
        CVec3 vecBodyDir = Player.m_vecBodyDir;
        vecBodyDir.z = 0;
        vecBodyDir.Normalize();
        float fScale = Camera.m_fWideGlance_CameraScale;
        pPhysic->m_vecNonInertialMovement += vecBodyDir * x * fScale;
        if(pPhysic->m_bIsStasisLikeMode)
        {
            // выводим объект из стазиса
            pPhysic->m_bIsStasisLikeMode = false;
            if(pPhysic->m_fIsInPreStasis())
                pPhysic->mfRemovePreStasis();
            else
                if(pPhysic->m_fIsInStasis())
                    pPhysic->mfRemoveStasis();
        }
    }
}

```

Обработка физики

Физика вызывается до игровой логики т.к. именно физикой устанавливаются текущие параметры физического состояния – лежит объект, летит или плавает. Если смещение объекта игрока рассчитывать во время обработки игровой логики (там, где работает код, специфичный для игрока), то из-за

ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ WIDEGLANCE В ИГРОВОМ ДВИЖКЕ

того, что физика объектов обрабатывается раньше, возникает задержка на кадр.

В физическом существе изменениям подверглась функция `mfCalculateMovement(float fDeltaT)` вычисляющая смещение объекта за указанный промежуток времени. Ранее для вычисления смещения использовалась текущая скорость и ускорение. Теперь в некоторых случаях добавляется текущее неинерционное смещение за кадр. **Жирным шрифтом** выделены добавления, связанные с неинерционным перемещением.

```
switch(m_nPhysicState)
{
case PAWN_PHYSIC_STATE_FREE_FALLING:
{
    // Свободное падение
    // Ничего не знаем о коллизиях
    CVec3 vecVelocity = m_vecVelocity;
    CVec3 vecForce = m_vecForce;
    CVec3 vecAcceleration = ( 1 / fMass)*vecForce;
    CVec3 vecNewVelocity=vecVelocity+vecAcceleration*fDeltaT;
    CVec3 vecPos = CollisionInfo.i.m_vecOrigin;
    CVec3 vecNewPos = vecPos +
        vecVelocity*fDeltaT +
        vecAcceleration*fDeltaT*fDeltaT/2.0f +
        m_vecNonInertialMovement;
    pCollisionInfo->m_vecTargetOrigin = vecNewPos;
    m_vecTargetVelocity = vecNewVelocity;
    return 1.0f;
}
case PAWN_PHYSIC_STATE_STEP_UP:
{
    CVec3 vecPos = CollisionInfo.i.m_vecOrigin;
    CVec3 vecNewPos = vecPos + fStepHeight * vecUp;
    pCollisionInfo->m_vecTargetOrigin = vecNewPos;
    m_vecTargetVelocity = m_vecVelocity;
    return 0.2f;
}
case PAWN_PHYSIC_STATE_STEP_FORWARD:
{
    CVec3 vecPos = CollisionInfo.i.m_vecOrigin;
    CVec3 vecNewPos = vecPos +
        m_vecHorizontalVelocity * fDeltaT +
        m_vecNonInertialMovement;
    pCollisionInfo->m_vecTargetOrigin = vecNewPos;
    m_vecTargetVelocity = m_vecVelocity;
    return 0.8f;
}
}
```

```

case PAWN_PHYSIC_STATE_STEP_DOWN:
{
    CVec3 vecPos = CollisionInfo.i.m_vecOrigin;
    CVec3 vecNewPos = vecPos - fStepHeight * vecUp;
    pCollisionInfo->m_vecTargetOrigin = vecNewPos;
    m_vecTargetVelocity = m_vecVelocity;
    return 1.Of;
}
case PAWN_PHYSIC_STATE_SLIDE:
{
    CVec3 vecVelocity = m_vecVelocity;
    CVec3 vecForce = m_vecForce;
    vecForce -= DotProduct(vecForce, vecUp) * vecUp;
    CVec3 vecAcceleration = ( 1 / fMass) * vecForce;
    CVec3 vecNewVelocity = vecVelocity + vecAcceleration * fDeltaT;
    CVec3 vecPos = pCollisionInfo->m_vecOrigin;
    CVec3 vecNewPos = vecPos +
        vecVelocity * fDeltaT +
        vecAcceleration * fDeltaT * fDeltaT / 2.Of +
        m_vecNonInertialMovement;
    pCollisionInfo->m_vecTargetOrigin = vecNewPos;
    m_vecTargetVelocity = vecNewVelocity;
    m_vecForce = vecForce;
    return 1.Of;
}
case PAWN_PHYSIC_STATE_STOP:
{
    pCollisionInfo->m_vecTargetOrigin = pCollisionInfo->m_vecOrigin +
    m_vecNonInertialMovement;
    m_vecTargetVelocity = m_vecVelocity;
    return 1.Of;
}
case PAWN_PHYSIC_STATE_MOVEUP:
{
    pCollisionInfo->m_vecTargetOrigin = pCollisionInfo->m_vecOrigin +
    CVec3(0, 0, fStepHeight) + m_vecNonInertialMovement;
    m_vecTargetVelocity = m_vecVelocity;
    return 1.Of;
}
default:
    return 1.Of;
}

```

Вообще говоря, в физике удалось ограничиться минимальными переделками. Помимо указанного места `m_vecNonInertialMovement` упоминается еще

в конструкторе (обнуление), при проверке на стазис (вместе со скоростью и приложенными силами) и в функции `mfEndPhysicalFrame` (обнуление в конце фрейма).

Такая реализация позволила свести к минимуму появление новых связей в проекте.

Не потребовалось создавать новый тип физики. Физическая библиотека по-прежнему не связана ни с рендером, ни с игровой логикой.

Почему разделена обработка физики и логики?

- во-первых, физическое поведение игровых объектов довольно однообразно. Движку *Shine* хватает всего трех видов физики для решения всех поставленных перед ним задач. Для специфической обработки физических взаимодействий (столкновение, входение в зону) физика предоставляет игровым объектам возможность подключить свои обработчики событий.
- во-вторых, перемещения объектов и проверка столкновений производятся попарно. То есть физика двигает сразу два объекта.
- за время физического фрейма объект может перемещаться несколько раз, взаимодействуя с несколькими объектами.
- Могут быть объекты, не имеющие физики, но имеющие логику и наоборот.
- Отделение физики от логики позволяет избежать размазывания кода физики по игровым объектам.

Прежде чем у физики ходящего существа появилось неинерционное перемещение, была предпринята попытка изготовить специальную физику для игрока. Для нее была бы переопределена функция определения смещения указанным выше образом. Но вычисление смещения пришлось бы вести тоже где-то в этой функции. Для этого пришлось бы организовать связь физики объекта игрока со встроенной в игрока камерой и логикой игрока (для определения направления взгляда). Возникли бы нежелательные лишние связи, которые усложняют систему, увеличивают время компиляции и чреваты возникновением ошибок. Тем более, что физический движок может находиться и в отдельном модуле, и даже вообще представлять в виде готовой библиотеки, разработанной сторонней командой.

*Для большинства приложений может хватить черырех-пяти физических моделей. Физика твердого тела, физика ходячего существа (симулирующая перешагивание через препятствия), физика механического устройства (двери, лифты), физика тряпичной куклы (*RagDoll*).*

Программирование оружия

Положение прицела на экране. Режимы наведения оружия

При смещении головы можно менять угол поворота оружия (речь идет не только о модели, но и о направлении вылета снарядов), а можно оставлять его направленным по нормали к экрану. Очевидный вариант изменения направления оружия – направление в центр экрана. При этом положение прицела остается неизменным, точка прицеливания располагается в центре поля зрения, что удобно. Неудобным в данной ситуации является то, что голова начинает конфликтовать с рукой – изменение направления выстрела происходит как при смещении головы, так и при смещении мыши. В конечном итоге это выражается в том, что в момент начала перестрелки игрок старается зафиксировать голову, переходя на «обычное» управление – «мышь + клавиатура». Единственное исключение из этого правила – огнемет. Наведение его при помощи головы может быть и не столь эффективно, но... понравилось большинству опрошенных игроков. Этот эффект, по-видимому, снижается при использовании усиленного неинерционного смещения, когда игрок испытывает непропорциональное смещение физического тела при смещении головы. В этом случае смещения головы вызывают сильные боковые смещения игрока. Конфликт не возникает потому, что контроль над положением головы включается не только во время перестрелки, но происходит постоянно. Если оставлять направление оружия неизменным, то возникает другая проблема – при наклонении монитора прицел смещается к низу или (что реже) к верху экрана. В центре экрана прицел будет размещаться только если игрок сидит прямо напротив монитора. Это может быть неудобным игроку, например, из-за бликов на экране. Можно рассмотреть вариант, когда прицел смещается только по горизонтали.

Направление вылета снарядов

Понятно, что снаряд, вылетевший из оружия, должен попасть в ту точку, которая расположена за прицелом. Недопустимы задержки на кадр.

Точка вылета снарядов

Точка, из которой вылетает снаряд, должна совпадать со стволом. Для случая, когда положение этой точки задается костью на модели оружия, никакие вычисления не потребуются. В противном случае нужна модификация соответствующего кода. Точка вылета снарядов должна находиться внутри коллизионного объема персонажа, иначе появится возможность стрелять из-за тонких стен или у игрока прижавшегося спиной к стене снаряды начнут взрываться в момент выстрела.

Положение модели оружия

Положение модели оружия. Камера не должна слишком далеко отдаляться от модели оружия (имеется в виду вид от первого лица). Камера не должна показывать оружие с обратной стороны (в целях оптимизации там могут быть удалены полигоны).

ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ WIDEGLANCE В ИГРОВОМ ДВИЖКЕ

Проблема наведения оружия (конфликт устройств ввода)

В случае, когда оружие наведено в центр экрана, возникает конфликт устройств ввода – оружием начинают управлять одновременно рука и голова. Этот конфликт смягчается в случае, если при смещении головы в стороны будет происходить сильное боковое смещение персонажа. Смещение головы выгодно использовать для управления боковыми перемещениями, появляется необходимость контроля за положением головы, а смещение прицела за счет сдвига персонажа несколько компенсируется поворотом направления взгляда в центр экрана.

Снайперский режим

Снайперский режим. При переходе в снайперский режим нужно сохранять направление прицеливания. Но возможно потребуется отключить/ограничить использование технологии.

Скорость полета снарядов при использовании неинерционного перемещения

В том случае если при смещении головы будет происходить непропорциональное смещение персонажа, возможна ситуация, когда персонаж сможет догнать (и перегнать) свои собственные снаряды. Нужно исключить возможность застрелить себя в спину. При сетевой игре возможны коллизии со своими снарядами на клиенте, когда снаряд рассматривается как препятствие, в которое попал игрок, и предпринимается попытка вытолкнуть игрока из снаряда. Визуально это воспринимается как дерганье камеры при пролете сквозь игрока своей ракеты.



1 Пропорциональное смещение объекта игрока.

2 Непропорциональное смещение объекта игрока.

При сильном смещении точка, в которую остается непрерывно наведенным оружие, оказывается на удобном расстоянии от игрока.

Этапы встраивания технологии в игровой движок

Предполагается, что поддержка устройства будет встраиваться в уже готовый игровой движок. Встраивание в движок Shine планировалось таким образом, чтобы обеспечить продвижение по возможности малыми шагами, имея возможность после каждого шага проверить работоспособность и применимость используемых алгоритмов. Затраты времени на реализацию каждого из описанных ниже этапов составят не более одного рабочего дня.

Шаги встраивания

- Прием данных с драйвера. В движке определяется класс, ответственный за связь с драйвером. Обработка ввода включается в игровой цикл.
- Для хранения координат глаза добавляется три новые консольные переменные – на координаты X, Y, Z. Применение консольных переменных открывает доступ к координатам головы из встроенного скрипта. Теперь координаты головы можно увидеть, проверив значение консольной переменной.
- Окончательное принятие решения о том, как будут использоваться координаты головы, будет ли изменяться матрица преобразования.
- В объект рендера добавляется вектор для хранения координат головы. Принимается решение об ориентации координатных осей для этих координат – они соответствуют системе координат камеры рендера. Нулевое положение соответствует положению головы, при котором видимая ширина экрана соответствует ширине поля зрения используемой при работе без устройства.
- В функцию приема данных с драйвера добавляется код, переводящий координаты камеры из системы, связанной с экраном (система драйвера), в систему координат камеры рендера.
- В рендер добавляется код подготовки к рендеру – вектор координат головы собирается из консольных переменных.
- В рендер добавляется код вычисления координат камеры с учетом координат головы. После этого перемещения головы начинают наглядно отображаться на экране. Но матрица еще не скорректирована. Производится проверка правильности преобразования координат – в ту ли сторону смещается камера.
- В рендер добавляется код коррекции матрицы преобразования.
- В рендер добавляется код коррекции пирамиды отсечения.
- Движковый и игровой код просматривается на наличие неучтенных механизмов отсечения, работа которых с новой матрицей преобразования может быть некорректной.
- Проверяется работа рендера – нормальное функционирование зеркал, порталов, окклюдеров, PVS, пирамиды отсечения,
- В класс – контроллер камеры добавляются переменные и код, необходимые для поддержки устройства. Флаг включения-выключения использования технологии. Виртуальный метод трассировки камеры для предотвращения залезания в стены. Масштаб камеры.

■ ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ WIDEGLANCE В ИГРОВОМ ДВИЖКЕ

- В рендер к коду, учитывающему положение головы, добавляется код трассировки, предотвращающей залезание камеры в стены.
- В контроллер камеры добавляется вектор, определяющий свободу перемещения камеры. Для контроллера камеры вида от первого лица оставляется только перемещение по вертикали – горизонтальное перемещение реализуется перемещением физического тела.
- В рендер добавляется функция вычисления направления на центр экрана при смещении головы.
- Модифицируется код оружия – при смещениях головы поворачивается и смещается оружие, изменяется направление вылета снарядов. Проверяются на играбельность варианты с изменением и сохранением направления вылета снарядов.
- Модифицируется код управления игроком – добавляется поворот вектора тяги при смещении головы – чтобы двигаться в направлении центра экрана. Проверяются на играбельность вариант с поворотом и вариант без поворота.
- В физику добавляется неинерционное смещение.
- Добавляется два новых управляющих воздействия – неинерционное смещение игрока вперед-назад и влево-вправо.
- В функцию приема данных с драйвера добавляется код посылы управляющих сообщений для неинерционного перемещения.
- Производится тестирование неинерционного перемещения. Проверяется скольжение вдоль стен, обратимость смещений, выход из стазиса, соответствие виртуального и реального масштабов перемещения.
- Проверка возможных способов увеличения ширины поля зрения.
- Проверка использования устройства в качестве джойстика. Увеличение масштаба неинерционного смещения.

Шаги, которые могут еще потребоваться

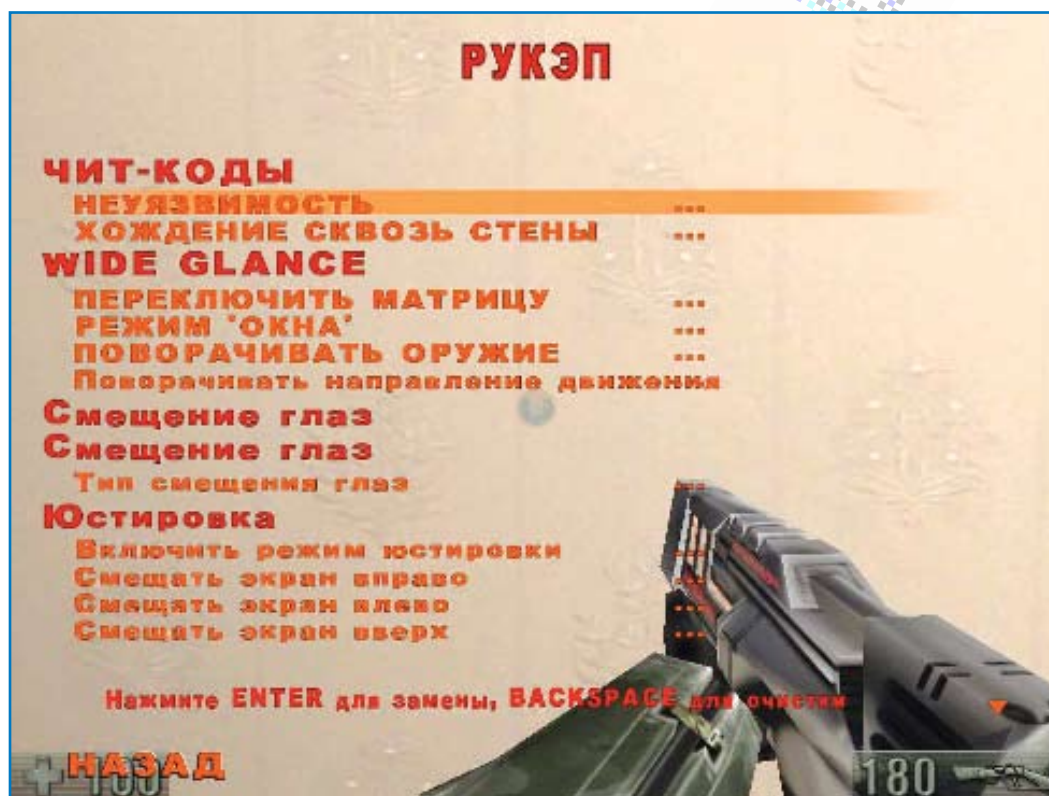
- Сетевая репликация. Требуется принятие решения о том, будет ли передаваться клиентам информация о перемещении головы игрока и в каком виде эта информация должна быть представлена.
- Доработка алгоритма перемещения камеры в виде от третьего лица.
- Управление камерой в стратегиях.
- Управление камерой в симуляторах (авиа и авто).
- 2D-интерфейс. Использование информации о перемещении головы в отображении интерфейса (наложение на элементы интерфейса envinmet-mapped текстур).

ПРИМЕРЫ

Игра «Жор-тур» (3D-движок “Shine”)

Меню настройки

Настройка управления



ЧИТ-КОДЫ

НЕУЯЗВИМОСТЬ – назначение кнопки, включающей/выключающей неуязвимость персонажа.

ХОЖДЕНИЕ СКВОЗЬ СТЕНЫ – назначение кнопки, включающей/выключающей проходимость персонажа сквозь стены.

WIDE GLANCE

ПЕРЕКЛЮЧИТЬ МАТРИЦУ – переключение между режимами с коррекцией матрицы преобразования и с поворотом камеры.

ПОВОРАЧИВАТЬ ОРУЖИЕ – переключение режима наведения оружия – по нормали к экрану или в центр экрана.

ПОВОРАЧИВАТЬ НАПРАВЛЕНИЕ ДВИЖЕНИЯ – переключает направление

■ ПРИМЕРЫ

движения – в центр экрана или по нормали к экрану.

СМЕЩЕНИЕ ГЛАЗ

Переключение модели, используемой для вычисления положения глаз.

ЮСТИРОВКА

Включить режим юстировки

Смещать экран вправо

Смещать экран влево

Смещать экран вверх

Смещать экран вниз

Смещать экран вперед

Смещать экран назад

Смещать глаз вправо

Смещать глаз влево

Смещать глаз вверх

Смещать глаз вниз

Смещать глаз вперед

Смещать глаз назад

Сохранить юстировку

Настройка параметров фильтрации



Настройка режимов отображения

**Консольные переменные и команды**

Консольные команды и переменные в движке Shine, относящиеся к технологии WideGlance. Изменяя значения некоторых этих переменных, можно ознакомиться с разными режимами использования технологии WideGlance.

`vr_WideGlanceEnable`

– общее включение-выключение использования технологии.

`vr_WideGlance_XPos`, `vr_WideGlance_YPos`, `vr_WideGlance_Zpos`
– позиция точки наблюдения относительно экрана.

Почему было сделано именно так. Информация о точке наблюдения передается рендеру при помощи трех консольных переменных. Этот нелогичный на первый взгляд шаг (можно было бы, конечно, и копировать напрямую) позволил вести встраивание технологии в движок параллельно с разработкой устройства и драйвера. И осуществлять отладки, эмулируя работу устройства изменением консольных переменных.

`vr_WideGlance_OriginalXPos`, `vr_WideGlance_OriginalYPos`, `vr_WideGlance_OriginalZPos`

– позиция излучателя относительно экрана. Значения этих переменных об-

■ ПРИМЕРЫ

новляются на каждом кадре. Значение этих переменных обновляется прямо из данных, получаемых от драйвера устройства, и к этим координатам не применяются никакие коррекции.

`vr_WideGlanceKeyBoardSpeed`

– множитель для управления скоростью перемещения камеры при клавиатурной эмуляции устройства. Переменная важна на начальном этапе подключения устройства – для клавиатурной эмуляции.

`vr_WideGlanceEyeOffsetX`, `vr_WideGlanceEyeOffsetY`, `vr_WideGlanceEyeOffsetZ`
– калибровочные значения – смещение глаз относительно излучателя ультразвука.

`vr_WideGlanceAdaptiveEyeOffset`

– переключение режимов коррекции положения глаз. Сферическая система координат, прямоугольная система координат, анатомическая модель.

`vr_WideGlanceScreenOffsetX`, `vr_WideGlanceScreenOffsetY`, `vr_WideGlanceScreenOffsetZ`

– калибровочные значения – смещение экрана относительно плоскости микрофонов.

`vr_WideGlanceSensorsBase`

– параметр устройства – расстояние между сенсорами

`vr_WideGlanceMonitorWidth`, `vr_WideGlanceMonitorHeight`

– параметры экрана

`vr_WideGlanceFOVScale`

– коэффициент увеличения ширины поля зрения. 1 – режим окна. Для 17-ти дюймового монитора ширина поля зрения 750 получается при значениях 2..3. Эту переменную можно плавно изменять из настроечного меню:

`vr_WideGlanceNoFOVChange`

– отключение влияния смещения головы по нормали к экрану. Переменная важна для определения требуемого в данном продукте режима отображения. В случае если эта переменная не равна 0, ширина поля зрения при положении глаза над центром экрана всегда будет равна ширине по умолчанию.

`vr_WideGlanceInvertPerspective`

– инвертирование перспективы (приближение головы к экрану сужает поле зрения). Переменная важна для определения требуемого в данном продукте режима отображения. При удалении головы от экрана будет происходить увеличение поля зрения.

`vr_WideGlance_StraightMovementScale`

– повышение чувствительности смещения физического объема по отношению к смещению камеры (головы) – составляющая перемещения «вперед-назад»

`vr_WideGlance_SideMovementScale`

– повышение чувствительности смещения физического объема по отношению к смещению камеры (головы) – составляющая перемещения «влево-вправо»

При значениях этих переменных равных 1 смещение головы будет эквивалентно отображаться в виртуальном пространстве. Значения переменных в диапазоне 1-15 обеспечивают комфортную работу с устройством. Эти переменные можно плавно изменять из настроечного меню.

При смещении головы по горизонтали подаются консольные команды

`WideGlanceStraightMove`

`WideGlanceSideMove`

Параметром этих команд является смещение головы в соответствующих направлениях. Величина этого смещения дается в относительных единицах. За 1 принимается расстояние от экранной плоскости до такого положения камеры, при котором FOV равно заданному по умолчанию. Для перевода смещения в игровую размерность используется хранящийся в контроллере камеры игрока масштабирующий коэффициент.

■ ПРИМЕРЫ

Демонстрационные уровни***Тестовый кубик***

Простой уровень для демонстрации технологии WideGlance. На уровне представлены объекты, полезные для проверок программной части в процессе встраивания технологии в игровой движок.

Зеркало.

Виртуальный экран с объектом за ним.

Масштабная линейка с шагом 10 см, для проверки соответствия масштабов реального и виртуального пространств.



Аквариум

Этот уровень служит для наглядной демонстрации возможностей «режима виртуального окна». Управление камерой: W, S – наклоны камеры вверх вниз. A, D – смещение камеры вверх вниз.



■ ПРИМЕРЫ

Гарнизон

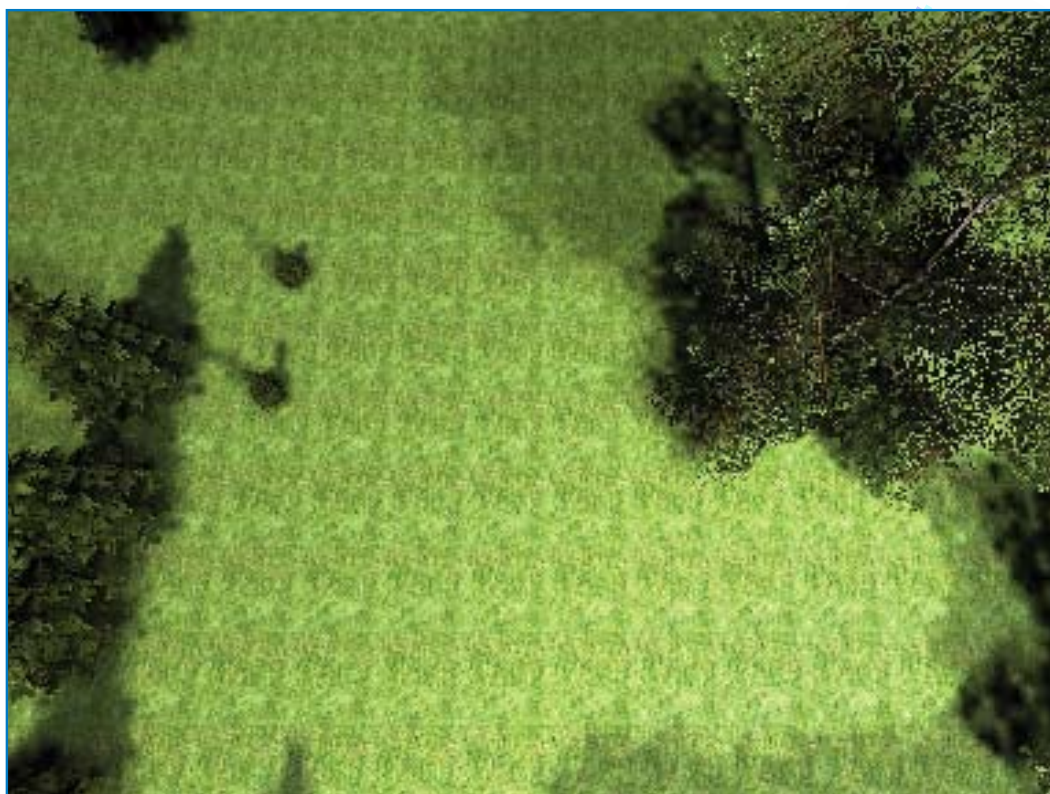
Демо-уровень игры «Жор-тур». Можно ознакомиться с различными режимами управления оружием и персонажем.



WORLD

Камера для стратегий

Уровень позволяет оценить использование технологии в режимах, когда камера находится на «высоте птичьего полета». Управление камерой : W, S – перемещение камеры вперед-назад. A, D – смещение камеры влево-вправо.



ПРИМЕРЫ

Игра “Space Invaders”

В этой игре координаты, получаемые с устройства ruCap, применяются для перемещения космического корабля влево/вправо и вперед/назад.

Используются функции:

LoadRucapDriver() – загрузка драйвера ruCap.

CloseRucapDriver() – выгрузка драйвера ruCap.

Из драйвера экспортируется функция GetHeadPos.

Обработка полученный координат производится в функции WinMain.



Игра “Lesson22” (Тир)

Пример представляет собой небольшую игру-тир, использующую OpenGL. В игре координаты, получаемые с устройства ruCap, применяются для позиционирования прицела.

LoadRucapDriver() – загрузка драйвера ruCap.

CloseRucapDriver() – выгрузка драйвера ruCap.

Из драйвера экспортируется функция GetHeadPos. Обработка полученных координат производится в функции Update.



■ ПРИМЕРЫ

Примерный код обработчика ввода

В файлах [CMyWideGlanceDriver.h](#) и [CMyWideGlanceDriver.cpp](#) содержится примерный код простейшего обработчика ввода с устройства.

Приводится код загрузки драйвера, опроса устройства, закрытия устройства.

Вы можете без каких-либо ограничений встроить этот код в свой проект, при необходимости дополнив его нужной вам функциональностью.

WIDEGLANCE

Определение направления взгляда (ориентация головы)

Устройство позволяет определить направление оси расположенного на голове излучателя, что с достаточной степенью точности может интерпретироваться как направление взгляда пользователя. Представляя ориентацию головы как тройку эйлеровых углов можно сказать, что устройство позволяет определить углы Yaw и Pitch. Угол Roll остаётся неопределённым. Представив ориентацию головы как тройку векторов Forward, Right, Up можно сказать, что устройство позволяет определить направление Forward.

Интерфейс

К вышеперечисленным функциям добавляется функция **wg_result_t GetHeadDirection(float& x, float&y, float& z)**

Функция возвращает направление оси излучателя (направление взгляда) в системе координат связанной с экраном. Вектор нормализован.

Параметры функции:

x – координата по оси X.

y - координата по оси Y.

z - координата по оси Z.

Возвращаемые функцией значения:

WG_RESULT_OK – значение возвращается при успешном получении и обработке координат.

WG_RESULT_FAIL – значение возвращаемое в случае ошибки.

Дополнительно в интерфейс будут включены функции для управления сглаживанием и фильтрацией. В настоящий момент ведётся проработка алгоритмов, и окончательный вид этих функций определится через некоторое время.

Используя получаемый при помощи функции GetHeadDirection вектор можно вычислить удобный Вам формат представления ориентации головы – эйлеровы углы, репер из трёх векторов, кватернион.

Точность определения ориентации головы.

Ожидаемая точность определения направления взгляда в пределах рабочей зоны – 5 угловых градусов.

Ожидаемая чувствительность устройства – 1 угловой градус.

Ожидаемая задержка сигнала – не более 0.25 сек.

Задержки сигнала и инертность устройства в определении направления взгляда будут несколько выше, чем при определении координат точки наблюдения.

Дополнение в Wideglance SDK после введения расчета вектора направления датчика

Приложения к пункту “**Описание датчика функций драйвера ruCap CMU-3**” (стр. 8).

Существующий комплект оборудования ruCap CMU-3 позволяет определять направление взгляда пользователя. Для использования данной возможности введены следующие функции:

wg_result_t GetAngles(&x,&y,&z);

Функция возвращает координаты вектора направления датчика излучателя. Вектор нормирован.

Функцию следует вызывать после вызова функции GetHeadPos.

В переменных x,y,z возвращаются координаты вектора.

Функция возвращает значение WG_RESULT_OK.

wg_result_t SetFilterParams(float fDynaMin,float fDynaMax);

Функция устанавливает минимальную и максимальную скорость движения головы пользователя. Данные скорости учитываются при расчете траектории движения головы пользователя. Возможные пределы для устанавливаемых значений от 0.2 м/с до 15 м/с. Устанавливаемый диапазон возможных скоростей влияет на чувствительность устройства к перемещениям головы пользователя. Т.е чем больше верхний предел скорости тем устройство более чувствительно.

fDynaMin – минимально возможная скорость движения головы пользователя.

fDynaMax - максимально возможная скорость движения головы пользователя.

Функция возвращает значение WG_RESULT_OK.

wg_result_t SetDefaultFiltration();

Функция устанавливает максимальную и минимальную скорость движения головы пользователя в значения по умолчанию. Значениями по умолчанию являются скорости 0.2 м/с и 15 м/с для минимально и максимально возможной скорости соответственно.

Функция возвращает значение WG_RESULT_OK.



ruCap

Лаборатория

124489, Москва, Зеленоград, Панфиловский проспект 10,
ОАО «НИИ Зенит», ООО «Рукэп», офис 413
Тел.: +7 (495) 982 5914

Офис в Москве

101000, Москва, ул. Мясницкая, д.24, стр.3, 5 этаж, оф. 2.
Тел./факс: +7 (495) 628 8347
Тел. +7 (495) 642 4252

Поддержка разработчиков: +7 (495) 642 4252

www.rucap.ru/develop

