



СМР

United Business Media

>> **ГОРЯЧАЯ ТЕМА: ИГРЫ ДЛЯ КАЗУАЛОВ**

(game)land

02(05), ФЕВРАЛЬ 2006

game**developer**

ВЕДУЩИЙ ЖУРНАЛ ОБ ИГРОВОЙ ИНДУСТРИИ **РОССИЯ**

>> **ДЕЛОВОЙ РАЗДЕЛ**

МАКСИМУМ ПОЛЬЗЫ ОТ
РУКОВОДСТВА

>> **ГЕЙМ-ШУИ**

КОГДА ОХОТНИК
СТАНОВИТСЯ ЖЕРТВОЙ

>> **ВНУТРЕННИЙ ПРОДУКТ**

МНОГОЯДЕРНЫЕ
ПРОЦЕССОРЫ

ПОСТ-МОРТЕМ
ПОБУЗИМ О
**GUITAR
HERO**





Привет!

В связи с внезапно наступившей GDC, некоторыми техническими сложностями наш выпуск был задержан. Однако, мы искренне верим в то, что это пауза была концентрацией перед решительным рывком вперед. С момента выхода последнего номера мы получили тонны писем с пожеланиями и предложениями, которые отразятся в содержании наших следующих выпусков. Кроме того, мы наметили планы долгосрочного сотрудничества со многими известными в индустрии людьми и компаниями. Предстоящий месяц будет полон важнейших событий и анонсов, а мы, в свою очередь, обещаем несколько приятных сюрпризов, которые вас вне всяких сомнений будут вам интересны и полезны. Встретимся в апреле!

Всегда ваши,
Анатолий Норенко, главный редактор,
Александр Логинов, зам. главного редактора

ОБЪЯВЛЕНИЕ

Корреспондент для журнала «Game Developer Россия»

Возраст: от 21 года

Образование: высшее или неполное высшее

Пол: не имеет значения

Профессиональные навыки: отличное знание русского языка, умение ясно излагать свои мысли

Качества: инициативность, ответственность, коммуникабельность, умение обучаться и совершенствовать навыки

Требования к кандидату:

- отличное знание российской игровой индустрии, хорошее знание игрового рынка в мире;
- быть в курсе всех новостей, событий, тенденций;
- умение собирать интересные факты и актуальную информацию;
- умение создавать аналитические материалы, всесторонне освещая проблему.

Резюме присылайте Анатолию Норенко на адрес editor@gamedeveloper.ru, указав в теме письма «Вакансия: корреспондент».

Журнал выходит 1 раз в месяц. №02 (05) ФЕВРАЛЬ 2006
[HTTP://WWW.GAMEDEVELOPER.RU](http://www.gamedeveloper.ru)

РЕДАКЦИЯ

Анатолий Норенко	editor@gamedeveloper.ru	главный редактор
Александр Логинов	aloginov@gamedeveloper.ru	зам. главного редактора
Александр Калинин	akalinin@gamedeveloper.ru	выпускающий редактор
Юлия Транквилицкая	proof@gamedeveloper.ru	литературный редактор / корректор

ART

Даниил Ткач	dtkach@gamedeveloper.ru	арт-директор
Дмитрий Бортоновский	dbortnovsky@gamedeveloper.ru	дизайнер

РЕДАКЦИЯ GAME DEVELOPER В США

Simon Carless	scarless@cmp.com	Editor
Jill Duffy	jduffy@cmp.com	Managing Editor
Brandon Sheffield	bsheffield@cmp.com	Assistant Editor
Cliff Scorso	cscorso@cmp.com	Art Director

РЕКЛАМНЫЙ ОТДЕЛ

По вопросам размещения рекламы обращайтесь, пожалуйста, к
Максиму Соболеву (sobolev@gameland.ru)

Телефон:	(095)935-7034;
факс:	(095)780-8824

ИЗДАТЕЛЬ И УЧРЕДИТЕЛЬ

Анатолий Норенко	spoiler@gameland.ru	издатель
Дмитрий Агарунов	dmitri@gameland.ru	генеральный директор

PUBLISHER

Anatoly Norenko	spoiler@gameland.ru	publisher
Dmitri Agarov	dmitri@gameland.ru	director

КАК СВЯЗАТЬСЯ С РЕДАКЦИЕЙ

Для писем: 101000, Москва, Главпочтамт, а/я 652, «Game Developer Россия»
E-mail: feedback@gamedeveloper.ru, Тел.: (095) 935-7034, Факс: (095)780-8824

Доступ в Интернет: компания Zenon N.S.P.
<http://www.aha.ru>; Тел.: (095)250-4629

За достоверность рекламной информации ответственность несут рекламодатели. Рекламные материалы не редактируются и не корректируются.

Редакция ждет откликов и писем читателей. Рукописи, фотографии и иные материалы не рецензируются и не возвращаются.

При цитировании или ином использовании материалов, опубликованных в настоящем издании, ссылка на «Game Developer Россия» строго обязательна. Полное или частичное воспроизведение или размножение каким бы то ни было способом материалов настоящего издания допускается только с письменного разрешения владельца авторских прав.

Game Developer Россия содержит материалы, лицензированные у CMP Media LLC. Указанные материалы переведены и опубликованы с разрешения Game Developer © 2006 CMP Media LLC. Все права защищены.

Game Developer Russia contains articles under license from CMP Media LLC. The articles are translated and reprinted by permission of Game Developer, copyright © 2006 CMP Media LLC. All rights reserved.

(game)land



WWW.GAMEDEVELOPER.RU



04 ПОСТМОРТЕМ GUITAR HERO

Когда за создание симулятора рок-гитариста берутся не просто опытные разработчики, но профессиональные музыканты, игроки могут быть уверены, что получат продукт отменного качества. Даниэл Зюсман и Грэг Ло-Пикколо рассказывают о том, как компания Harmonix осуществила свою давнюю мечту – дала возможность выразить себя в музыке тем, кто отродясь не держал гитары в руках.

09 ПОЛОЖЕНИЕ В ИНДУСТРИИ ИГРЫ ДЛЯ КАЗУАЛОВ

Игры на обеденный перерыв? Интернет-игры? Даже те, кто издает и разрабатывает эти игры, по сей день не определились, как их называть. Рынок казуальных игр давно перестал быть «казуальным»: сегодня освоение новых платформ оказывается прибыльным бизнесом для предприимчивых дельцов.

14 ОГРОМНЫЕ ВОЗМОЖНОСТИ OGRE ИНТЕРВЬЮ С «ЛЕСТОЙ»

Разработка своего движка – это почти всегда непозволительная роскошь. Выбор чужого – далеко не простая задача. Проект «Стальные Монстры» наглядно показал, что замечательные по качеству игры совсем не обязательно делать на коммерческих движках.



27 ОТ ЗАДУМКИ К ПОБЕДЕ ЖАК X: COMBAT RACING

Осенью 2004 года студия Naughty Dog решила сделать игру в относительно новом для себя жанре. Ричард Леманчард рассказывает, как компания смогла, несмотря на все трудности, справиться с задачей.



КОЛОНКИ

18 АКУСТИЧЕСКАЯ ФИКСАЦИЯ

АВТОР: ДЖЕСС ХАРЛИН
Контроль качества аудио

[ЗВУК]

21 ДОЗА ПИКСЕЛЕЙ

АВТОР: СТИВ ТЕОДОР
Гонки на выживание

[АРТ]

36 ВНУТРЕННИЙ ПРОДУКТ

АВТОР: РУСЛАН АБДИКЕЕВ [ПРОГРАММИРОВАНИЕ]
It has begun...

19 ДЕЛОВОЙ РАЗДЕЛ

АВТОР: КЛАРАНДА МЕРРИПЕН
Максимум пользы от руководства

[БИЗНЕС]

24 ВНУТРЕННИЙ ПРОДУКТ

АВТОР: МИК ВЕСТ [ПРОГРАММИРОВАНИЕ]
Многоядерные процессоры

[ПРОГРАММИРОВАНИЕ]

42 ВНУТРЕННИЙ ПРОДУКТ

АВТОР: АЛЕКСЕЙ БОРЕСКОВ [ПРОГРАММИРОВАНИЕ]
Программирование современных GPU

20 ГЕЙМ-ШУЙ

АВТОР: НОЙ ФАЛЬШТАЙН
Когда охотник становится жертвой

[ДИЗАЙН]

32 ДОЗА ПИКСЕЛЕЙ

АВТОР: АНДРЕЙ АКСЕНОВ
Шейдинг в «Магии крови»

[АРТ]

46 УПРАВЛЕНИЕ

АВТОР: АЛЕКСАНДР ЛОГИНОВ [МЕНЕДЖМЕНТ]
Наши разработчики зарубежом

[МЕНЕДЖМЕНТ]

48 ЗА СЕМЬЮ ПЕЧАТЯМИ

АВТОР: АЛЕКСАНДР КАЛИНИН
Обзор литературы

[ОБУЧЕНИЕ]

РАЗДЕЛЫ

02 ГЛАЗА НА ЭКРАН Разработка мобильного контента, успехи Nintendo DS и другое...
50 ДОРОЖЕ ТЫСЯЧИ СЛОВ Outpost Kaloki X

[ГЛАЗА НА ЭКРАН]

ЕСТЬ НОВОСТИ? ПОДЕЛИТЕСЬ С НАМИ: EDITOR@GAMEDEVELOPER.RU

В ЦЕНТРЕ ВНИМАНИЯ ПРОШЕДШЕЙ CES БЫЛА
РАЗРАБОТКА МОБИЛЬНОГО КОНТЕНТА

ОСТАЛЬНОЕ ПУБЛИКУ НЕ СЛИШКОМ ИНТЕРЕСОВАЛО

КОГДА-ТО CES (CONSUMER ELECTRONICS SHOW) СЧИТАЛАСЬ ЛУЧШИМ В МИРЕ МЕСТОМ ДЛЯ ПРЕМЬЕР ВИДЕОИГР. Но после учреждения E3 в 1995 году работники индустрии стали уделять ей значительно меньше внимания.

Хотя CES с тех пор и не считается «всемирной ярмаркой игровых инноваций», как однажды окрестил ее журналист Билл Канкел (Bill Kunkel), в этом году ей удалось преподнести пару сюрпризов. Помимо HD-DVD аддона к Xbox 360, больше всего хочется отметить организованную Digital Hollywood серию докладов Game Power. Среди вызвавших наибольший интерес тем – повышение стоимости разработки, вопросы кроссплатформенной переносимости и рост рынка мобильных и казуальных игр.

«Мы сталкиваемся с замечательными прототипами игр, геймплей которых весьма инновационен. Но в конце концов приходим к выводу, что такие игры не будут продаваться и поэтому не издаем их, – отметил президент издательства Ubisoft Джей Кохен (Jay Cohen). – Так больше продолжаться не должно. Нам бы хотелось, чтобы эти игры увидели свет. Мы

не можем продолжать выпускать блокбастеры с бюджетом в 50 миллионов долларов». Кохен признал, что сервис Xbox Live Arcade от Microsoft будет способствовать созданию обширного рынка для творческих малобюджетных игр для Xbox 360.

Учитывая тот факт, что приставки и мобильные телефоны становятся все более популярными, актуальность вопросов переносимости резко возрастает. Они доставляют немало головной боли разработчикам, стремящимся делать игры более чем для одной платформы.

«Мы издали мобильный вариант NFL, а для того, чтобы сделать его по возможности совместимым с большим числом платформ, нам пришлось обратиться к трем разным разработчикам для получения трех различных версий игры, – рассказал вице-президент по продажам и маркетингу компании Jamdat Mobile Минард Гамильтон (Minard Hamilton) во время второй серии докладов.

Джейсон Рубинштейн (Jason Rubinstein) из компании Motorola также говорил об огромных масштабах рынка мобильных игр: «Во всем мире активно



Более чем 150 тысяч человек посетили CES, проходившую с 5 по 6 января

используется примерно 300 миллионов поддерживающих игры мобильных телефонов. Побольше, чем 12-13 миллионов проданных PSP? Средняя цена игры для мобильного телефона составляет где-то около \$4.55, что сопоставимо с прокатом фильма. В то же время игра для PSP стоит от 40 до 50 долларов. Таким образом, до тех пор, пока контингент потребителей мобильных игр будет весьма разнообразным, рынок сохранит свои масштабы».

Фрэнк Сифалди (Frank Cifaldi)

NINTENDO DS ЗАВОЕВЫВАЕТ ЯПОНСКИЙ РЫНОК

В ЭТОМ ГОДУ ВО ВРЕМЯ ПРАЗДНИКОВ В ЯПОНИИ НАРАСХВАТ ШЛА ВОВСЕ НЕ СТАРТОВАВШАЯ ПО ВСЕМУ МИРУ XBOX 360, А САМАЯ МАЛЕНЬКАЯ В МИРЕ КАРМАННАЯ КОНСОЛЬ – NINTENDO DS. Двухэкранное устройство, кажется, и в самом деле выходит на новые рынки. Похоже, начинает сбываться прогноз Сатору Ивата (Satoru Iwata) о расширении игрового ассортимента за счет выпуска большого количества хардкорных игр, который он дал в сентябре прошлого года на Tokyo Game Show.

Частично благодаря высочайшему качеству и широкому выбору программного обеспечения за первые 13 месяцев было продано более пяти миллионов Nintendo DS – лучший показатель в Японии. Для достижения аналогичного уровня продаж Game Boy Advance потребовалось 14 месяцев, а PlayStation 2 – 17.

Незадолго до Рождества продавалось примерно полмиллиона DS в неделю – на 400 процентов больше, чем PSP. К Новому году почти все консоли были проданы. После этого последовали официальные извинения от

Nintendo и обещания в кратчайшие сроки выпустить на рынок новую партию устройств. А так как время поджимало, а найти DS в Японии было практически невозможно, то цены выросли почти вдвое. Подобное явление наблюдалось в США и Великобритании в период нехватки Xbox 360.

Громадным успехом в Японии DS обязана оригинальному и тщательно дифференцированному программному обеспечению. Уже как минимум четыре игры превысили миллионный уровень продаж: Nintendogs, Animal Crossing: Wild World и головоломки Brain Age и Brain Flex. Трех играм для DS – Nintendogs, Mario Kart DS и Mario 64 DS также сопутствовал большой успех на Западе – их продажи в США перевалили за миллионный рубеж. Западный рынок все еще остается благоприятным для консольных игр. Несмотря на то, что позиции PSP там весьма прочны, большинство сходится во мнении, что у DS потенциал ничуть не меньше.

Брендон Шеффилд (Brandon Sheffield)

ФЕСТИВАЛЬ НЕЗАВИСИМЫХ ИГР – 2006

ОРГАНИЗАТОРЫ ФЕСТИВАЛЯ НЕЗАВИСИМЫХ ИГР (ПРОВОДИМОГО CMP GAME GROUP, ИЗДАЮЩЕЙ ЖУРНАЛ GAME DEVELOPER) ОБЪЯВИЛИ ФИНАЛИСТОВ 2006 ГОДА. ПОБЕДИТЕЛИ БУДУТ НАЗВАНЫ В МАРТЕ НА GDC В САН-ХОСЕ.

Всего номинировалось 118 игр – конкуренция была жесточайшей. Сорок судей Фестиваля должны были определить наиболее достойные и оригинальные. В этом году обновленное жюри состояло из профессиональных разработчиков из крупных студий (Nihilistic, Vicarious Visions, Shiny, Criterion/EA, Activision и Big Huge Games), журналистов основных игровых сайтов, посвященных «маленьким» играм (TIGSource, GameTunnel и DIYGames), и создателей игр-победителей прошлых конкурсов (Gish и N).

Финалисты, претендующие на Seumas McNally Grand Prize и \$20 000:

- экшн-стратегия Darwinia от Introversion;
- французская стратегическая онлайн RPG Dofus;
- забавная головоломка Professor Fizzwizzle от Grubby Games;
- инновационный космический тайтл Weird Worlds: Return To Infinite Space от Digital Eel и симулятор экосистемы WildLife Tycoon: Venture Africa от компании Pocketwatch Games.

Стоит отметить финалистов в номинации «За инновации в геймдизайне». Это уже упомянутая выше Darwinia, Rumble Box; Strange Attractors, управляемая одной кнопкой; Braid, за которой можно просто приятно провести время; приключение с неплохим сюжетом The Witches Yard. Кроме того, в номинации «За лучшую игру для интернет-браузера» финалистами были объявлены Dodge That Anvil от Looney Tunes (в прошлом Shockwave), химическая головоломка Moleculous и бесстыжая задиристая игра Dad 'N Me.

Список остальных финалистов:

Номинация «За технологическое совершенство»:

Saints & Sinners Bowling,

Tribal Trouble, Tube Twist, Darwinia, Crazy Ball.

Номинация «За художественные инновации»:

Dofus, Darwinia, Putt

Nuts, Glow Worm, Thomas And The Magigcal Words.

Номинация «За аудио инновации»:

Professor Fizzwizzle, Saints & Sinners Bowling, Dodge That Anvil, Glow Worm, Weird Worlds: Return To Infinite Space.

Общий призовой фонд конкурса, победителями которого становились такие игры как Wik & The Fable Of Souls, Alien Hominid, Oasis и Gish, в этом году составляет \$35 000. Из них \$20 000 предназначено для будущего обладателя Seumas McNally Grand Prize. По \$2 500 получают победители в других номинациях, которые будут определяться среди всех финалистов. Финалисты в номинациях «За лучший мод» и «За лучший студенческий дебют» будут объявлены позднее (см. www.igf.com).

Саймон Карлесс (Simon Carless)

Претенденты на SEUMAS MCNALLY GRAND PRIZE

Return To Infinite Space от компании Digital Eel – это оригинальная и короткая игра для PC, в которой игроку предстоит совершить нелегкое космическое путешествие. Также ему придется хорошенько попотеть в яростных схватках с инопланетянами.



Игра WildLife Tycoon: Venture Africa от компании Pocketwatch Games представляет собой увлекательный симулятор африканской экосистемы. Игроку нужно найти компромисс в использовании пищи и воды, разводить животных и вообще испытать все прелести «дикой» жизни.



Darwinia от Introversion – стильная ретро стратегия-экшн, разработанная британскими самоучками. Недавно ее начали распространять через систему Steam от Valve.



Dofus компании Ankama – самобытная онлайн-игра, сделанная по технологии Flash. Разработчики использовали популярную концепцию «стратегия – RPG» и превратили проект в продолжительный мультяшный поиск магических яиц дракона.



Professor Fizzwizzle от Grubby Games – хитроумная 2D головоломка, повторяющая классический стиль The Incredible Machine: в ней больше чем 230 уровней и имеется встроенный полнофункциональный редактор.



POSTMORTEM

ПОБУЗИМ С GUITAR HERO ОТ HARMONIX

КОМПАНИЯ HARMONIX MUSIC SYSTEMS БЫЛА ОСНОВАНА АЛЕКСОМ РИГОПУЛОСОМ (ALEX RIGOPULOS) И ЭРАНОМ ЭГОЗИ (ERAN EGOZY) В 1995 ГОДУ. АЛЕКС И ЭРАН ПОЗНАКОМИЛИСЬ В M.I.T MEDIA LAB, И ИХ ПЕРВОНАЧАЛЬНОЙ ЦЕЛЬЮ БЫЛО ПРИ ПОМОЩИ ТЕХНИКИ ПОМОЧЬ НЕ-МУЗЫКАНТАМ ИСПЫТАТЬ ВСЕ УДОВОЛЬСТВИЕ ОТ МУЗИЦИРОВАНИЯ.

Этим принципом продиктованы все наши девелоперские усилия и по сей день. Мы все считаем, что играть музыку — одно из самых захватывающих и упоительных занятий, недоступное, однако, большинству людей в силу временных затрат и усидчивости, требующихся, чтобы достичь мастерства в игре на каком-либо инструменте. Мы были убеждены, что если нам удастся снять технические барьеры при игре на музыкальных инструментах, мы подарим нашей аудитории небывалые возможности, а сами сможем зарабатывать на этом. Прошло некоторое время, прежде чем мы сообразили, что видеоигры — прекрасная платформа для претворения в жизнь нашей давней мечты. И как только кусочки мозаики в наших головах сложились воедино (чему способствовало знакомство с Parappa the Rapper), мы ее больше не разбирали.

Сообразно нашей цели в своей стратегии мы сосредоточились на музыкальных продуктах. Начав с Frequency и Amplitude для PlayStation 2, мы выпустили затем серию Karaoke Revolution сразу на нескольких платформах. По пути мы немного отвлечлись на EyeToy: AntiGrav, но это совсем другая история. Из разработки каждого тайтла мы извлекли ценные уроки, и когда на идейном горизонте замаячил Guitar Hero, уже были во всеоружии. Guitar Hero оказался самым успешным воплощением устремлений Harmonix на сегодняшний день. Мы получили огромное количество отзывов от музыкантов и обычных игроков, которые восхищались, насколько точно Guitar Hero передает ощущения игры на электрогитаре. На пике азарта Guitar Hero ломает грань между видеоигрой и игрой на гитаре. Этот проект — наша большая гордость, учитывая изначальную уверенность в том, что когда-нибудь мы это сделаем, то время и те усилия, которые мы затратили на достижение нашей цели.





Иллюстрация Клифа Скорсо (Cliff Scorso).
С фотографии Алиссы Шайнзон (Alyssa Scheinson).

ДОСЬЕ



ИЗДАТЕЛЬ: RedOctane

РАЗРАБОТЧИК: Harmonix Music Systems

ДАТА ВЫХОДА: Ноябрь 2005 г.

ЧИСЛО РАЗРАБОТЧИКОВ (МАКС.): 49

СРОК РАЗРАБОТКИ: Девять месяцев

ПЛАТФОРМЫ: PlayStation 2

ОБЪЕМ ПРОЕКТА: 303,000 строк кода
32,100 строк скриптов

В ХОДЕ РАЗРАБОТКИ РАЗБИТО ГИТАР: 97

В ХОДЕ РАЗРАБОТКИ СОЖЖЕНО ГИТАР: 1

ДЭНИЭЛ СУССМАН

(Daniel Sussman), продюсер из Harmonix Music Systems, играет на гитаре уже 17 лет. Его любимые инструменты – 1990 Gibson Les Paul и 1967 Gretsch Astro-Jet

ГРЕГ ЛО-РИКОЛЛО (Greg

LoPiccolo) – руководитель проекта Guitar Hero, присоединившийся к Harmonix в 1998-м. Его нынешняя бас-гитара – 1974 Fender Precision с примочками от Seymour Duncan. Он по сей день не прошел Cowboys From Hell на сложности Expert, чего невероятно стыдится. Присылайте ваши комментарии к статье на editor@gamehug.com

POSTMORTEM

ИЗ НИЧЕГО

Возможность создать продукт уровня Guitar Hero была для Harmonix неожиданностью. RedOctane вышли на нас как раз в тот момент, когда у нас была свободная команда. А когда мы все хорошенько обдумали, то поняли, что именно такую игру и мечтали создать. Нам выделили скромный бюджет и всего 9 месяцев на разработку, но у нас уже была солидная программная база, богатые наработки по дизайну, а также небывалый энтузиазм и рвение к работе в данном жанре.

Руководитель проекта Грэг Ло-Пикколо (Greg LoPiccolo) и старший звукорежиссер Эрик Брозиус (Eric Brosius) играли вместе (басист и гитарист соответственно) в Tribe, выдающейся бостонской рок-группе далекого прошлого. Арт-директор Райан Лессер (Ryan Lesser) исколесил США, будучи гитаристом The Laurels. Программист игровых систем Дэн Шмит (Dan Schmidt) – фронтмен и ритм-гитарист группы Honest Bob and the Factory-to-Dealer Incentives, играющей в стиле инди-поп. А продюсер Дэниэл Зюсман (Daniel Sussman) в данный момент является гитаристом Acro-Brats, панк-группы из Бостона. Многие другие члены студии были или остаются членами всевозможных музыкальных коллективов. Рок – это значимая часть нашей жизни и часть культуры нашей компании. Guitar Hero стал для нас отличной возможностью отплатить добром (не говоря уже о возможности здорово повеселиться) музыке, которую мы любим.

ГДЕ НЕ ПРОГАДАЛИ

1. Прочная солидарность в видении темы

Когда мы начинали работу, единое видение будущей игры было очень важным. Ведь первоначально RedOctane просто поинтересовались возможностью сделать проект с использованием гитары. После первого 10-минутного мозгового штурма мы решили, что это будет игра об электрогитаре. Единогласную поддержку по-

лучила неограниченная насыщенность роком: им должен был быть пропитан любой элемент игры – от подборки музыки до визуального оформления и дизайна интерфейса. Когда возникал какой-то вопрос по арту или дизайну, мы просто задумывались: «А это зажигательно?» – и принимали решение в соответствии с этим. Вся команда была солидарна с таким видением, благодаря чему мы могли сосредоточиться на работе и сэкономили кучу времени,

не устраивая никчемных споров. Также мы очень признательны RedOctane, которые разделили наш взгляд и предоставили простор для его реализации. Они присутствовали на обсуждении ключевых рабочих вопросов, но в целом нам доверяли и позволили сделать то, чего нам хотелось.

2. Внушительный этап препродакшна

Мы весьма экстравагантно начали первый этап работы: создали весь руководящий состав и провели ликбез по року. У брата Грэга



Эти поломанные гитары от сторонних производителей пали в неравном бою

отличная квартира с проекционным телевизором, укрепленным на потолке, и грандиозной стереосистемой. Вот там мы и собрались попить пивка и посмотреть клипы. Часа три мы смотрели концерты Led Zeppelin, AC/DC, Rolling Stones и прочие классические записи. Быть может, это покажется надуманным, однако то, что мы собрались и обсудили, каким великим был Джимми Пейдж, насколько кто любил или не выносил The Who, отлично сплотило команду. Здорово было выяснить, что все мы – упертые музыкальные снобы. Вот после этого мы и приступили к работе, набрасывая персонажи, места действия и понемногу собирая играбельную версию. Такой продуктивный период препродакшна во многом способствовал успеху Guitar Hero. Мы сами были удивлены, как много из первоначальных концептуальных набросков вошло в итоге в финальную версию игры.

3. Мощный первый билд

Еще со времен Amplitude у нас были многодорожечные аудиофайлы «Dope Nose» от Weezer, и мы решили, что их вполне можно использовать для тестовой версии, поскольку там было гитарное соло. Программист Дэн Шмит на скорую руку прикрутил простенький 2D-интерфейс с белыми линиями, бегущими вниз по экрану. После этого он дописал простую систему подсчета очков... а вслед за тем вся команда начала бить рекорды один за другим. Эрик Брозиус (Eric Brosius), директор по звуку, сделал в домашней студии еще несколько песен (он написал некие подобию «Walk This Way», «Back in Black», «Ain't Talkin' 'Bout Love») – и мы подсели. То, что Guitar Hero оказался очень увлекательным еще в самой ранней версии, придало нам уверенности в том, что мы работаем над успешным проектом.

4. Мощная программная база

Еще одной причиной, по которой мы собрали первую версию Guitar Hero так скоро, был плод нашего пятилетнего труда – отлично сконструированный и отлаженный движок для музыкальных игр. На самом деле мы не использовали ни фрагмента синхронизирующего кода из наших ведущих тайтлов – просто те уроки, которые мы усвоили в работе над ними, сильно упростили нам программирование игрового ядра. Системы игровых арен и персонажей, разработанные для Karaoke Revolution, послужили основой



для аналоговичных систем в Guitar Hero. Мы бы ни за что не выпустили игру за девять месяцев, не будь у нас такого продвинутого и гибкого игрового движка и кода. Они не только оказались отличным подспорьем в разработке, но и добавили нашей работе гибкости. Мы могли создавать различные механизмы геймплея за несколько дней вместо нескольких недель. Также мы активно черпали бонусы из полученного ранее опыта. У нас за плечами уже было несколько трехмерных ритм-экшн игр, что помогло нам избежать некоторых подводных камней.

5. Удачные гитары

Работая над игрой, мы знали, что ее успех или провал во многом зависит от качества контроллеров-гитар. Это нас очень сильно напрягало, поскольку в разработке периферийных устройств мы ничего не смыслили. К великой чести RedOctane они сумели разработать и выпустить контроллеры, которые превосходили все наши ожидания. Вскоре после начала разработки RedOctane запросили у нас список пожеланий по части технического исполнения гитары. Мы пожелали сенсор угла наклона и вибратор еще задолго до того, как вообще получили представление о том, как они будут использоваться в игре. Нам ответили, что это резко повысит стоимость производства, но мы заявили, что нам нужны эти фишки, потому что они реально будут жечь, хоть мы и не могли объяснить почему. И RedOctane согласились, за что огромное им спасибо!

ЧТО ПОШЛО НЕ ТАК

1. Не было гитар

Производство контроллеров стартовало одновременно с разработкой. В результате в работе над игрой большую часть времени использовались гитары сторонних производителей, которые мы еле-еле – и в дефиците – сумели отыскать через Интернет. Все это были устройства низкого качества, а перебрали мы их незнамо сколько. До сих пор у нас в кладовке лежит груда пластиковых инструментов. У этих гитар были дурацкие кнопки, не было никакого вибратора, и рычажок, отвечающий за удар по струнам, работал только в одном направлении, что не позволяло нам протестировать великое множество наших фишек. И для настройки сложности они не годились. А наш первый контроллер от RedOctane мы получили всего за несколько дней до E3. И даже после E3 они поставлялись нам буквально по одной, так что нам не хватало гитар на каждого разработчика, и мы вечно слонялись по студии, одалживая друг у друга контроллер. К тому же мы прекрасно понимали, что отдел тестирования должен работать с гитарами как можно дольше, что еще сильнее сокращало наш гитарный пул. В общем, недостаток гитар сильно осложнил нам тестирование как игры, так и оборудования.

2. Режим Freestyle

Нам не терпелось реализовать в игре фристайл-режим, который позволил бы игроку сочинять свои сумасшедшие соло, используя все те залихватские гитарные выкрутасы, которые мы все так любим. Мы вложили в реализацию этой возможности много

усилий и времени, но в итоге, как это ни печально, вынуждены были от нее отказаться. Это была очень амбициозная задумка, и у нас банально не было времени на отстройку звучания и грамотную интеграцию режима в геймплей. Некоторые из нас считают, что игра стоила свеч, другие с ними не согласны.

3. Несоблюдение графика

В общем и целом нам успешно удалось построить четкий график и придерживаться его практически все время. Однако в процессе работы всплывало множество на первый взгляд тривиальных штук (вроде призовых товаров в магазине, вступительного ролика, заставки при успешном прохождении), время на которые либо было неадекватно оценено, либо не было учтено в графике вовсе. А между тем они отъедали весомую долю ресурсов. Впервые мы это осознали на подступах к бета-версии, когда продолжали как сумасшедшие доливать в игру все новый контент. Если бы не наша образцовая команда – не избежать бы нам проблем. Это типичные грабли для разработчиков, и мы прежде уже на них наступали. Мы думали, что усвоили урок и выстроили необходимую структуру, чтобы предотвратить эту проблему, поэтому столкнуться с ней снова было крайне неприятно. Как следствие этого, мы в Harmonix установили куда более суровые и конкретные правила для разработчиков и вспомогательных продюсеров, которые распространяются на все наши проекты.

С тех пор мы разработали шаблон графика для всех своих проектов, со списком общих фишек, которые необходимо реализовать на каждом этапе. Разумеется, у каждого проекта есть свои специфические особенности и много уникального контента, но теперь мы с куда меньшей вероятностью упустим какую-нибудь рядовую фишку, стандартную для всех наших тайтлов. И общий курс разработки – когда реализовывать ту или иную деталь геймплея – стал куда очевиднее (в качестве примеров можно привести систему сохранения-загрузки для карт памяти и соответствующие индикаторы, готовые и реализованные в игре победные заставки и многое прочее).



В данной ситуации было бы здорово использовать вибратор

POSTMORTEM

Все эти вещи – не из области квантовой физики, и их довольно просто воплотить. Но стоит проглядеть несколько таких мелочей – и можно здорово погореть, особенно если команда работает в жестком графике. Перерабатывать никому не хочется, и хотя нам в Harmonix не удалось этого избежать, мы делаем все возможное, чтобы свести вероятность подобных форс-мажоров к минимуму. Недостаток времени ввиду неграмотного планирования – это клеймо, которое мы всеми силами стараемся с себя смыть. Если бы мы более полно отразили перечисленные мелочи в планах, нам удалось бы реализовать их быстрее и, быть может, выгадать время на те фишки, которые нам пришлось зарубить. Отсюда понятно, что...

4. Нам пришлось зарубать фишки

Хорошие фишки. Такие, как режим тренировки, который позволил бы выбирать отдельные фрагменты композиций и замедлять их, чтобы было легче заучить. Мы собирали много фокус-групп и заключили, что большинство игроков справляется с игрой без специального обучения. Сделав такой вывод, мы решили отказаться от тренировочного режима, хотя многие впоследствии сильно на это сетовали.

Необходимость вырезать некоторые фишки была отчасти продиктована философией нашей работы. Мы изначально составили очень амбициозный список фишек, уже тогда закладываясь на то, что нам скорее всего придется здорово его сократить. Мы брались за разработку глубокой и насыщенной, по нашему мнению, игры, но по ходу дела сужали размах, чтобы успеть вовремя. Мне кажется, мы отбросили верные фишки, хотя расставаться с некоторыми все равно было тяжело.

5. Ни одного трека от AC/DC

Ну и пожалуйста, не очень-то и хотелось... Хотя мы и довольны итоговым списком композиций для игры, все песни, о которых мы грезил, нам получить не удалось. Мы располагали внушительными средствами на лицензирование, однако некоторые суммы превышали наши возможности, а некоторые команды попросту не были заинтересованы в сотрудничестве. Не станем перечислять их – они и так знают, о ком речь. Быть может, удастся получить их работы в сиквеле – мы не оставляем на это надежды.

ВРЕМЯ ИМЕЕТ ЗНАЧЕНИЕ

Мы считаем Guitar Hero своим большим достижением, и ничего бы не вышло, если бы мы твердо не вознамерились сделать его в отведенное время. Игровые проекты Harmonix, как правило, имеют сжатые сроки разработки и много неясностей в дизайне, с которыми порой не так-то просто разобраться. За последние несколько лет мы выработали стратегию выживания в таком жестком ритме, и наши навыки здорово облегчили нам работу над Guitar Hero. Вот наиболее примечательные из них:

– Прозрачные коммуникационные барьеры между всеми сферами и уровнями менеджмента вкупе с агрессивной политикой по упрощению коммуникации. На работе любой может обсуждать с кем

удобно все что пожелает: во время летучек, на встречах, за обедом. Ясно ведь, что каждый член команды в какой-то мере несет ответственность за итоговый результат – вне зависимости от его роли. Поступает множество предложений и жалоб, большинство отклоняется, но удачные идеи в итоге находят свое воплощение.

– Большой внутристудийный отдел тестирования. Как правило, у нас слишком мало времени, чтобы всецело полагаться на издательский отдел контроля качества. Нам нужна оперативная работа с каждым новым билдом и постоянная связь отдела тестирования с разработчиками для быстрого выявления и устранения багов. И на старте каждого проекта мы предполагаем, что львиную долю тестирования будет осуществлять наш отдел. Мы активно вовлекаем наших тестеров в процесс создания игры, они становятся со-вестью всей разработки. На данный момент мы не видим разработку игр без подобной степени вовлеченности в нее тестеров.

– Ранние первые билды, много раз пересмотренные и переделанные. На старте разработки полновесный диздок – это не панацея, поскольку пока толком не знаешь, что из отраженного в нем действительно будет цеплять. Отличным примером в случае Guitar Hero служит интеграция фишек Star Power и вибрато. Мы перепробовали уйму эффектов модуляции и игровых воплощений, прежде чем добрались до релизного варианта. Сейчас тогдашнее решение кажется очевидным, но тогда оно не являлось таковым вовсе. Потребовалось несколько месяцев проб и постоянных пересмотров, прежде чем итоговый дизайн был утвержден.

Ввиду сжатых сроков мы вынуждены были строго следить за тем, чтобы наш проект не разрастался. В этом нам помогало общее убеждение (выстраданное в процессе работ над прежними продуктами), что если девелоперский ресурс ограничен, необходимо сосредоточить его весь в ядре проекта, а не размениваться на факультативные ответвления. И то мы придерживались этого принципа со скрежетом зубным: все равно потратили время на фишки, которые в итоге не выгорели. Нам пришлось вырезать некоторые детали, с которыми не хотелось расставаться; однако общую рок-н-рольную атмосферу, которую мы берегли как зеницу ока, удалось сохранить.

ПОКАЖИ-КА ИМ, ДРУЖОК

Мы, создатели Guitar Hero, до сих пор поражены, что за эту игру нам еще и заплатили деньги. Несмотря на некоторые нервные моменты, работать над проектом было сущим удовольствием – от начала до конца. Выход игры в продажу помог нам сделать поразительное и вместе с тем приятное открытие: столько людей по всему миру грезят о славе рок-звезд, как и мы! И неважно, что ты воображаешь себя патлатым гитаристом, держа в руках всего лишь пластиковый контроллер, – ведь в какой-то мере он все равно делает тебя таковым. Мы научились по-новому веселиться, зажигать, отбрасывать все сомнения, пока работали над этой игрой. И нам не терпится применить полученные знания в нашем следующем проекте. Подсказка: белых ремней и подвешенных над поясом гитар в нем не будет.



Motion capture
Harmonix
заказывали в
House of Moves



>> ПОЛ ХИМАН

Photo: Derek Jones and Elena Buetler

ПОЛОЖЕНИЕ В ИНДУСТРИИ: ИГРЫ ДЛЯ КАЗУАЛОВ

ИГРЫ ДЛЯ КАЗУАЛОВ? НАРОДНЫЕ ИГРЫ? КОМПАКТНЫЕ ИГРЫ?

Игры на обеденный перерыв? Интернет-игры? Даже те, кто разрабатывает и издает эти игры, по сей день не определились, как их называть. Однако, перефразируя Джастиса Поттера Стюарта, который не сумел дать определение порнографии, мы скажем: «Увидев, мы их точно узнаем».

Как бы то ни было, если текущие тенденции сохранятся, то вскоре станет нелегко даже распределить по категориям эти крохотные увлекательные загружаемые игры, которые являются главной опорой игровых порталов и производителей мобильных устройств. Казуальные игры набирают вес, становятся более насыщенными, съедают большие бюджеты, а некоторые уже продаются по \$30 – то есть дороже, чем так называемые «потребительские» консольные игры.

Разумеется, сам рынок Casual-игр давно перестал быть «казуальным»: сегодня освоение новых платформ оказывается прибыльным бизнесом для предприимчивых дельцов.

Студия PopCap Games из Сиэтла, традиционно считающаяся ведущим разработчиком в данной нише, сейчас подписывает контракт на разработку контента для развлекательных и интерактивных телесистем, установленных в самолетах. Портят даже автоматы типа «однорукий бандит».

«Все больше устройств приспособивается к установке игр, и наиболее успешными на них являются самые незамысловатые забавы... в то время как более привычные нам хардкорные игры не находят там достаточно широкой аудитории, – утверждает Джеймс Гверцман (James Gwertzman), директор по развитию бизнеса в PopCap. – У игр для Casual-игр широкая область применения, они привлекательны тем, что могут просочиться на какую угодно платформу – подобно воде, затекающей в любую трещинку».

Гверцман говорит, что он готов играть на повышение на этом рынке, который, по его прогнозам, будет очень быстро развиваться. Да и не оправдан ли риск? Ведь достаточно взглянуть на перечень 10 самых популярных игр на мобильных порталах, чтобы понять, что практически все они – Casual-игры. Действительно, одна из наиболее

ПОЛ ХИМАН

(Paul «The Game Master» Hyman) был главным редактором GamePower от CMP Media. Сейчас он ведет колонку, посвященную видеоиграм, в The Hollywood Reporter. Пол пишет об играх уже более десяти лет. Адрес для связи – phyman@gdmag.com



ПОЛОЖЕНИЕ В ИНДУСТРИИ: ИГРЫ ДЛЯ КАЗУАЛОВ

популярных – это Bejeweled 2, последняя вариация флагмана от PopCap, который положил начало жанру «рядостроек» и на сегодняшний день насчитывает 6 миллионов зарегистрированных копий.

Недавнее исследование агентства DFC Intelligence (Сан-Диего) показывает, что казуалы из одной Северной Америки в 2005 году потратили \$241 млн. на закачиваемые игры, а к 2009 году истратят еще \$1,7 млрд. Алексис Мэдригал (Alexis Madrigal) из DFC Intelligence говорит по этому поводу: «Определенно, у рынка игр для казуалов огромный потенциал. Сейчас компании, занимающиеся казуальными играми, получают в год от \$10 млн. до \$20 млн. дохода, однако через несколько лет некоторые станут зарабатывать по \$100 млн. и даже больше».

РАЗРАБОТЧИК – ВСЕМУ ГОЛОВА

Сайт PopCap Games (www.popcap.com) дает представление о типичной бизнес-модели производителя игр для казуалов. Ассортимент из 25 бесплатных браузерных игрушек (которые зарабатывают за счет рекламы) для затравки к тому, что PopCap называет Deluxe-версиями игр. И хотя некоторые люди так и продолжают играть в «халяву», в которой всего по несколько уровней, изображение и звук пониженного качества, однако срок игры не ограничен, – они всегда могут скачать полноценную игру и поиграть в нее час или немного больше. По истечению этого времени игра перестает функционировать и предлагает пользователю снова разблокировать ее, уплатив около \$20.

«Мы наконец видим, что индустрия начала потихоньку экспериментировать с различными ценами и ознакомительными периодами, – говорит Гверцман. – Однако в целом бизнес-модель осталась прежней».

Стандартная для индустрии «доля перехода» (то есть процент пользователей, которые опробовали игру, а затем ее купили) составляет от 0,1% до 1,5% для обычной игры и 1,5%–2,5% для высококлассного продукта. Средняя доля для игр PopCap – 2%.

Понятно, что с таким процентом рынок игр для казуалов строится на крупных тиражах. Если разработчик потратил на производство игры несколько сотен тысяч долларов, а затем 98% сыграв-



Джон Уэлч, президент и исполнительный директор издательства PlayFirst

ших не стали платить, и только 2% раскошелились на «зарегистрированную версию», то очевидно, что аудиторию необходимо значительно расширить. Добиться этого можно несколькими способами.

Первый способ, разумеется, – заработать безупречную репутацию, чтобы потребители выстраивались в очередь за вашей продукцией и забыли об играх конкурентов. По оценке Гверцмана, за пять лет 25 игр от PopCap были скачаны 150 млн. раз – и каждый тайтл принес прибыль. «Многие из них стали хитами», – утверждает Джеймс, уточняя, что под хитом подразумевает казуальную игру, проданную тиражом более 100 тысяч копий. По его подсчетам, в 2004 году на долю PopCap приходилось 15% рынка игр для казуалов.

Гверцман добавил, что более чем рад содействовать разработчикам, которые готовы к новшествам: «Мы выпустили наш движок с открытым кодом PopCap Engine (<http://developer.PopCap.com>) – тот самый, на котором сами делаем игры. Любой разработчик может бесплатно скачать его и использовать. Наша программа поддержки девелоперов не предусматривает плату за лицензирование движка. Просто помяните нас в титрах».

Джеймс пояснил, что выпуск движка с открытым кодом – лучший способ побудить небольшие студии к творчеству и снять с них непосильную задачу производства собственного движка.

«Мы просто даем им наш движок, – говорит Гверцман, – и пускай сосредоточатся на разработке хороших игр вместо того, чтобы изобретать колесо».

Рецепт хорошей игры, по словам Гверцмана, – забыть про жесткое расписание и месяцами напролет создавать опытные образцы, прежде чем приступить непосредственно к производству.

«Мы потому и выпустили так мало проектов, что большинство наших идей еще не реализовано полностью, – поясняет Джеймс. – Все почему-то считают, что нужно бросить публике горсть игр – и посмотреть, за какую уцепятся. Мы считаем, что нет ничего вреднее для индустрии, чем засилье посредственных продуктов».

Лучшие игры от PopCap Джеймс описывает как «отлично расслабляющие и забавные». «Люди играют в них, чтобы развлечься, а не потягаться с другими или получить заряд адреналина. Боль-

Bejeweled 2 от
PopCap



шинство хардкорных геймеров просто не понимает игр для казуалов. Например, в Bejeweled 2 есть режим, где невозможно проиграть: просто нет условия проигрыша – и все тут. Но и не в уходе от проигрыша смысл игры – он просто в том, чтобы расслабиться», – поясняет Гверцман.

Пример другого подхода к рынку игр для казуалов являет собой компания PlayFirst из Сан-Франциско.

ИЗДАТЕЛЬ ВАМ В ПОМОЩЬ

Менее двух лет назад, в апреле 2004-го, Джон Уэлч (John Welch) и его партнер Брэд Эделман (Brad Edelman) решили, что пора рынку игр для казуалов получить своего первого полнофункционального издателя.

«Разработка игр становилась все более затратной, распространение также усложнялось, появилась возможность выпускать игры для разных платформ в нескольких странах и в локализованных версиях, да и деловая сторона вопроса – финансы, маркетинг, продажи, распространение, юридические тонкости – все это стремительно развивалось», – говорит Джон Уэлч, президент и соучредитель компании. – Маленькие команды были плохо приспособлены к преодолению всего этого, что отвлекало их от того, что они действительно умели и чем жаждали заниматься, – от разработки игр».

В частности, Уэлч упомянул, что год назад разработка одной игры обходилась в \$50-75 тысяч, а нынче обходится уже в \$100-150 тысяч: «Вот почему игры наподобие нашего бестселлера Diner Dash подняли планку по части новшеств, глубины и внешнего вида».

Да и объемы дистрибутивов растут как на дрожжах: современные игры с трудом умещаются в 10Мб.

PlayFirst не только финансирует производство, а затем ведет продукт на различных платформах, но также специализируется на подписании контрактов с интернет-порталами вроде Yahoo!, AOL, MSN Games, что стимулирует распространение игр.

«Представьте себе, – предлагает Джон, – разработчик заключает соглашение только с PlayFirst, а мы уже организуем всю дистрибуцию».

Как правило, издатель старается заключить соглашения на максимально выгодных условиях с наибольшим количеством порталов, которые получают процент с продаж, а затем делит прибыль с разработчиком. В разных случаях роялти может очень сильно отличаться.

«Мы постоянно работаем над перечнем типов дохода, который получаем от продажи игр, стараемся завязывать более прочные отношения и заключать более крупные сделки. Доход, таким образом, получается выше, чем если бы мы просто швыряли нашим партнерам пачку игр на продажу», – говорит Кенни Динкин (Kenney Dinkin), вице-президент и исполнительный продюсер PlayFirst.

В ведении Динкина – шесть игр из портфолио PlayFirst, а также выбор, какими проектами компания будет заниматься в дальнейшем. Этакая комбинация науки и искусства, по его собственным словам.

«Многим нашим игрокам за сорок, многие из них – женщины, и мы стараемся, чтобы наш ассортимент учитывал специфику рынка», – рассказывает Кенни. – К примеру, Diner Dash с его героиней Фло, которая бросает работу и открывает собственное дело, а затем выстраивает целую ресторанный империю, показалась нам очень свежей. Ее авторы, GameLab, проделали колоссальную работу, скрепив увлекательную тему и захватывающий игровой процесс, за которым можно проводить часы. Для нас это был отличный дебют. Он действительно отвечал критериям игр, которые нам интересны».

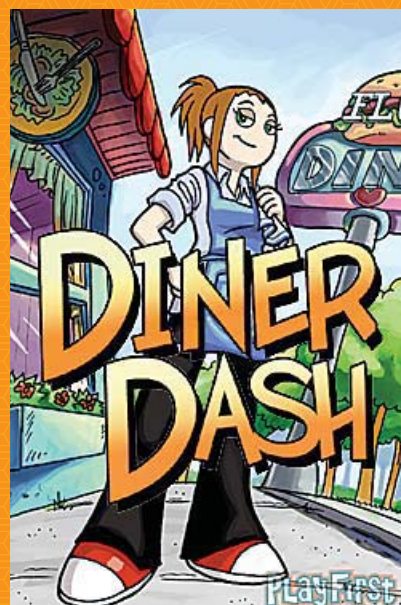
«Принимается игра или отклоняется, целиком определяет план по обновлению ассортимента. Проект должен относиться к требуемой категории, будь то игра в слова, стратегия, экшн или пазл», – поясняет Динкин. – Нам нужны игры с очень простой механикой, доступной рядовому пользователю. Однако оригинальная идея, которая, на наш взгляд, возымеет успех, – что-нибудь необычное, не посредственное, – привлечет наше внимание и финансы независимо от того, вписывается она в рамки определенных нами категорий или нет».

Однако Кенни предостерегает от чрезмерной оригинальности: «Если продукт выйдет чересчур самобытным и массовый рынок не сможет его переварить, он сильно рискует остаться без аудитории вне зависимости от раскрученности портала, на котором продается».

Тем не менее Джон Уэлч заявляет, что всегда находится в поиске талантов: «Если вы – лихой программист, художник, продюсер или дизайнер; если вы жаждете покинуть студию из сотни работников и влиться в команду из трех человек, где ваше слово действительно будет веским; если у вас достаточно ресурсов на создание независимой компании, которая предложит PlayFirst что-то стоящее, то 2006 год – отличное время, чтобы это сделать».

ПИШЕМ «ПОРТАЛЫ – «ПРОДАЖИ» В УМЕ

Выбор стоящих игр для Yahoo! Games – одного из крупнейших порталов, на который ежемесячно заходит 23 миллиона геймеров, – это «своего рода магия», по словам Тома Козика (Thom Kozik), старшего менеджера по развитию бизнеса. На Yahoo! Games сейчас хранится более 230 игр для казуалов, и еженедельно поступает 12-15 новых предложений от разработчиков и издателей. Из них при-



Diner Dash, разработка GameLab



Кенни Динкин, вице-президент PlayFirst



ПОЛОЖЕНИЕ В ИНДУСТРИИ: ИГРЫ ДЛЯ КАЗУАЛОВ

нимается только два-три, и одновременно столько же игр покидает текущую базу. Двери Yahoo! открыты для всех, кто ни поступится.

«Мы берем те игры, которые кажутся нам лучшими, играем в них – и выбираем самые захватывающие, – поясняет Козик. – Звучит слишком упрощенно, но мы не можем себе позволить отталкивать пользователей прямо у витрины. Каждую неделю они приходят к нам, чтобы увидеть самые лучшие игры, сыграть в некоторые и, быть может, их купить. И если они обнаружат хлам, они пойдут куда-нибудь еще. Мы, конечно, могли бы расширить каталог, но вместе с тем возросли бы и шансы отыскать в нем низкопробный продукт».

Весь фокус в том, говорит Козик, чтобы выбрать те игры, которые найдут положительный отклик именно в аудитории Yahoo! Games, 80% которой – люди в возрасте от 21 до 60. И 45% – женщины.

«Ассортимент у нас довольно обширный, – говорит Козик. – Части нашей публики нравятся игры одного типа – например «рядостройки» вроде Bejeweled, а некоторые постоянно пробуют что-то новое, во что они прежде не играли».

Том признает, что за исключением нескольких эксклюзивных игр на Yahoo! Games размещены те же продукты, что и на прочих

порталах, однако добавляет, что дела у Yahoo! идут значительно лучше благодаря его способности к «персонализации» услуг.

«Все основано на предположении, что какая-то игра наверняка вам понравится, если вы уже приобретали что-то подобное, – поясняет Козик. – Мы уже некоторое время ведем рейтинги пользователей в нашем музыкальном разделе и недавно прикрутили ту же систему к игровой части. Игроки формируют рейтинги игр, которые им нравятся и которые нет, а мы исходя из этого определяем, что им еще предложить. Например, если вы приобрели Bejeweled 2, мы предложим вам еще несколько подобных игр».

Также Yahoo! обогащает игровые возможности другими способами. Игроки могут состязаться с друзьями в браузерных играх, участвовать в онлайн-турнирах, которые проводит Yahoo!, читать

обзоры казуальных игр и скачивать их.

Мы выяснили, что обилие игровых возможностей под одним брендом очень способствует его привлекательности в глазах пользователей. Это позволяет нам достигать результата порядка 7 миллиардов минут непрерывной онлайн-игры на портале в месяц».

Разумеется, Yahoo! дороже покупает игры, которые свободно интегрируются во всю сеть Yahoo! Games. Как правило, условия договора с издателем или разработчиком строятся на разделе прибыли, которая зависит от отношений Yahoo! с компанией-партнером и с самой игрой.



Fate, разработка WildTangent

«Обо всем можно договориться, – поясняет Козик, – но если игра умеет передавать очки прямо в профиль пользователя на Yahoo!, где он может затем их увидеть, и вообще хорошо вписывается во всю систему, то мы будем более щедрыми, поскольку ценим это».

Том говорит, что в последнее время определение «игра для казуалов» распространяется на все более сложные жанры. В качестве примера он приводит Fate от WildTangent, «незамысловатую игру в стиле RPG». От нее никто не будет ожидать того же, что от «заряженного» World of Warcraft, – она как раз из того сорта «продвинутых» казуальных игр, на которых Yahoo! Games пробует взимать по \$30 за регистрацию вместо обычных \$20.

«Игры наподобие Fate мы используем, чтобы расширить спектр наших предложений: ставим высококлассный продукт на одну полку с ширпотребом, – говорит Том. – Нам кажется, что потребитель понемногу отходит от концепции «один товар на все случаи жизни», и нами был отмечен значительный рост продаж премиум-игр: пользователи видят их более высокое качество и богатые игровые возможности. Тот же Fate, к примеру, способен увлечь вас на месяц, а то и больше».

Другая причина более высокой цены за премиум-продукт, по словам Козика, состоит в том, что разработчик и издатель хотят возместить сравнительно высокие затраты на разработку. Если стандартная «рядостройка» стоит от \$50000 до \$150000, то бюджет Fate, уверяет Козик, вполне мог составить около \$1 млн. На самом деле Fate обошелся WildTangent в \$250000 (по словам генерального директора компании Алекса Джона (Alex St. John)).

КАЗУАЛЬНЫЕ ИГРЫ ДЛЯ ПРИСТАВОК

На рынке игр для казуалов Microsoft вряд ли можно назвать новичком. На портале MSN Games нашли пристанище 120 наименований, в то время как аудиторию самой популярной категории игр на портале – пазлов – на все 70% составляют женщины.



Том Козик, старший менеджер по развитию бизнеса Yahoo! Games

«Рынок игр для казуалов всего за несколько лет из ничего вырос до оборотов в сотни миллионов долларов, – говорит Грег Канесса (Greg Canessa), управляющий Xbox Live Arcade. – Если верить IDC, к 2009 году его оборот составит \$900 млн.»

Твердо вознамерившись перенести толику этого успеха в область рынка видеоигр, в ноябре 2004 года Microsoft запустила Xbox Live Arcade – и очень скоро поняла, что у нее в руках золотая жила.

«Доля перешедших с бесплатного ознакомительного режима к зарегистрированному нас поразила, – рассказывает Канесса. – Для рынка загружаемых PC-игр эта доля составляет в среднем менее 1%, среди наших же 20 игр процент перехода был порядка 12%. Хотя наша изначальная база игроков составляла всего несколько сот тысяч, и несмотря на то, что с точки зрения конечного пользователя Xbox Live Arcade была далеко не совершенна, мы считаем, что достигли успеха».

Канесса не скрывает недовольства некоторых пользователей: «Если в приставку не был вставлен диск из базового комплекта, нельзя было ни скачать новые игры, ни разблокировать их, ни даже сыграть в те, что уже загружены на жесткий диск. К тому же требовалось уплатить \$50 за пользование Xbox Live. Бесплатного обслуживания предусмотрено не было».

Год спустя, одновременно с запуском приставки нового поколения от Microsoft, стартовала и доработанная версия системы Xbox 360 Live Arcade – и в ней, по словам Грэга, удалось преодолеть все возникавшие прежде трудности. Диск больше не требуется, система полностью интегрирована в приставку, изначально доступна и абсолютно бесплатна. Канесса утверждает, что через постоянный менеджмент ассортимента игр в Live Arcade он всеми силами содействует расширению ее аудитории.

«На запуске новой консоли первые пять миллионов приставок предназначены для хардкорных игроков, готовых по три дня в снегопад стоять в очередях и платить по \$500 за то, чтобы первыми получить заветный девайс. А потому 70–80% первоначального ассортимента игр мы решили заточить под хардкорную аудиторию. Это не значит, что там будут сплошь и рядом шутеры класса Halo. Просто мы подобрали игры, которые имеют смежную привлекательность и будут интересны всем, даже опытным геймерам.

По мнению Грэга Канесса, успех – в правильной подборке игр из нескольких жанров. В результате среди 10 продуктов, подготовленных к запуску Live Arcade, оказались три казуальные игры (Bejeweled 2, Hexic HD и Zuma), пять классических тайтлов (Joust, Gauntlet, Geometry Wars Retro Evolved, Mutant Storm Reloaded и Smash TV), стратегия Outpost Kaloki X и бильярд Bankshot Billiards 2.

«Наша система куда масштабнее обычных интернет-порталов, – поясняет Канесса, – и жанровое разнообразие наших тайтлов богаче, нежели на большинстве казуальных геймерских ресурсов».

Грег сообщает, что в неделю получает порядка десяти новых предложений от студий и издательств, и рассчитывает на возрас-



Грег Канесса,
управляющий
Xbox Live Arcade.

тание этого количества с десяти по состоянию на прошедший ноябрь до тридцати пяти к грядущему лету.

«Поскольку у нас игр меньше, чем на казуальных PC-порталах, нам нужно внимательно следить за ассортиментом, – говорит Канесса. – Мы должны поддерживать высокий уровень качества, обеспечивать выбор игр для широкой аудитории, а также избегать дешевых поделок. Наша задача вовсе не в том, чтобы вывалить на прилавок четыреста тайтлов – такой избыток не пойдет на пользу ни нам, ни разработчикам, ни потребителю».

Итак, качество игр для казуалов растет, а параллельно снижается шанс небольших команд на выпуск настоящего хита. И все же на этом развивающемся рынке по-прежнему остается множество возможностей заявить о себе.



PlayFirst's TriJinx.

>> ОПЫТ ПИТЕРСКОЙ СТУДИИ «ЛЕСТА»

ЗАДАВАЛ ВОПРОСЫ: АЛЕКСАНДР КАЛИНИН

Буду рад всем вашим вопросам и комментариям. Пишите мне на akalinin@gamedeveloper.ru.

ОГРОМНЫЕ ВОЗМОЖНОСТИ OGRE: МИФ ИЛИ РЕАЛЬНОСТЬ?

ОПЫТ ПИТЕРСКОЙ СТУДИИ «ЛЕСТА»



РАЗРАБОТКА СВОЕГО ДВИЖКА – ЭТО ПОЧТИ ВСЕГДА НЕПОЗВОЛИТЕЛЬНАЯ РОСКОШЬ. ВЫБОР ЧУЖОГО – ДАЛЕКО НЕ ПРОСТАЯ ЗАДАЧА. У качественных коммерческих движков есть один важный ограничитель – цена. Среди множества бесплатных движков большинство не подходит для серьезной разработки. Бытует мнение, что на бесплатном движке вообще невозможно создать ничего по-настоящему привлекательного.

Но консерватизм хорош только в разумных пределах. Проект «Стальные Монстры» наглядно показал, что замечательные по качеству игры совсем не обязательно делать на коммерческих движках. На вопросы GDR о движке и об игре в целом отвечали Сергей Титаренко, руководитель проекта «Стальные Монстры», и

«почти главный» программист Сергей Кузнецов. Вот что нам удалось узнать.

ПОЧЕМУ OGRE?

Q: Очень многих волнует вопрос выбора движка. Однако разработчики никогда подробно не аргументируют свой выбор. Поэтому, если можно, попробуйте одним предложением ответить на вопрос – почему OGRE?

A: В основном потому, что по соотношению цена/качество у него мало конкурентов.

Q: Да, в данном аспекте он, пожалуй, лучший в мире. Какие вообще движки – платные/бесплатные – предлагались сначала?

A: Изначально предполагалось использовать движок собственной разработки. Но не сложилось...

Q: Поподробнее не расскажете?

A: На разработку новой архитектуры практически не было времени. Нужно было сразу начинать наполнение игры контентом, а не заниматься движком.

Q: Рассматривался ли движок Irrlicht? Два слова о нем.

A: Да, он рассматривался в числе прочих. На данный момент он являлся бы неплохой альтернативой, но тогда OGRE казался более совершенным.

СНАЧАЛА НЕЛЕГКО

Q: Как долго осваивали движок? Были в команде сотрудники, знакомые с ним до разработки? Проводились какие-либо специальные внутренние лекции/обсуждения по его использованию?

A: Освоение OGRE не представляло трудности – у него прекрасная документация. Первые опыты с ним начались уже через несколько минут после скачивания архива с исходниками. Сотрудников, ранее имевших дело с OGRE, на начало разработки не было, но это также не вызвало затруднений: освоение произошло быстро и практически безболезненно для всех программистов – как для тех, кто начинал работу над проектом, так и для тех, кто присоединился позже. Специальных лекций или семинаров по OGRE не проводилось, но каждый из нас всегда мог обратиться за советом или помощью к тому человеку, который лучше разбирается в той части движка, по которой возникал вопрос.

Q: Были проблемы с компиляцией движка/SKD? Приш-



лось ли столкнуться с чем-нибудь особенным?

А: Первоначально были затруднения, но не с самим OGRE, а с STLPort. Все проблемы исчезли, когда мы отказались от iostreams из STLPort и стали использовать потоки из родной версии STL нашего компилятора.

Q: Каким компилятором вы пользовались?

А: MSVC 7.0

Q: Общались с другими командами, работающими с OGRE?

А: Систематического общения не было, но иногда мы случайно встречались на различных форумах. Или не случайно — когда дело происходило на форуме ogre3d.org.

Q: Насколько помог ресурс <http://www.ogre3d.org/wiki/>? Приходилось входить в контакт с разработчиками движка?

А: На начало разработки игры OGRE Wiki просто не существовал, а потом, по мере знакомства с исходниками, уже возникало намного меньше вопросов. Хотя некоторые статьи из раздела Tutorials на официальном сайте OGRE, несомненно, были весьма полезны. Самим же разработчикам OGRE мы своими проблемами не докучали — документации было более чем достаточно.

Q: Насколько сильно изменился сам OGRE во время разработки? Эти изменения как-нибудь повлияли на работу? С какими трудностями из-за этого пришлось столкнуться?

А: Первоначально мы пытались вносить изменения из главной ветки OGRE в нашу

версию, но потом, в связи с большими расхождениями, этот процесс стал слишком трудоемким, и мы от него отказались. Наши же патчи вряд ли были бы внесены в главную ветку, поскольку большинство из них были грубыми хаками, призванными увеличить функциональность или быстродействие даже ценой потери переносимости и архитектурной красоты. В частности была разрушена совместимость нашей версии OGRE с Linux, MacOS и OpenGL. В результате мы пришли к собственной версии OGRE, работающей исключительно через DirectX и только под Windows, базирующейся на OGRE 0.14 с некоторыми бэкпортами из 0.15 и множеством собственных изменений.

ЦИКЛ РАЗРАБОТКИ

Q: Какие форматы моделей/анимации использовались при разработке?

А: OGRE .mesh и собственный формат для описания анимации.

Q: Как осуществлялся их импорт в движок?

А: Модели экспортировались утилитами с сайта ogre3d.org, анимация настраивалась собственными инструментами.

Q: С помощью каких инструментов разрабатывались уровни? Почему?

А: Уровни создавались в редакторе на базе игрового движка с рендерингом с помощью OGRE. Это давало возможность сразу увидеть уровень таким, каким он будет в игре.

Q: Что скажете о физике в «SM»?

А: Физика писалась самостоятельно — в связи со специфическими требованиями к ней в проекте. Нужно было обеспечить взаимодействие большого количества объектов и реализовать более-менее реалистичное поведение самолетов при приемом быстродвижении.

Q: Пожалуйста, подробнее расскажите про collision detection. Вы пользовались, например, OgreOpcode? Насколько это оказалось удобно?



>> ОПЫТ ПИТЕРСКОЙ СТУДИИ "ЛЕСТА"

А: Мы не использовали ни OP CODE, ни ODE, поэтому не можем дать им оценку. Система обнаружения столкновений в Pacific Storm – нашей собственной разработки, но она нас вполне удовлетворяет.

Q: Расскажите о звуке. OGG или MP3? Какие библиотеки использовались?

А: Для воспроизведения музыки в формате ogg мы использовали audiere, для прочих звуков – собственную обвязку для DirectSound.

Q: Ваша оценка FMOD и audiere?

А: Мы не использовали FMOD ни в одном из наших проектов, поэтому здесь просто нечего ответить. А audiere вполне достойно справляется со своими обязанностями.

Q: Как было реализовано использование шейдеров в движке?

А: Шейдеры довольно широко используются в игре, например специальным пиксельным шейдером производится текстурирование ландшафтов. Большинство шейдеров написано на Cg и подключается посредством системы материалов OGRE. Исключение – шейдеры для отображения поверхности одного из вариантов воды, которые основываются на сторонней разработке (были безвозмездно предоставлены автором) и читаются из файлов эффектов HLSL.

КОНКРЕТИКА. КАК БЫЛИ РЕАЛИЗОВАНЫ СРЕДСТВАМИ ДВИЖКА...

Q: ...LOD?

А: Где было возможно, использовалась встроенная LOD-система OGRE. Сетки для LOD'ов создавались художниками, а не ге-



«Воспроизводится с разрешения Lesta Studio».

нерировались автоматически. В некоторых случаях LOD'ы переключаются программно – например, на большом удалении от камеры модели кораблей и самолетов заменяются простыми сетками, в то время как на ближних расстояниях они имеют множество отдельных движущихся частей.

Q: ...небо?

А: Текстурированная полусфера, перемещающаяся вслед за камерой. Солнце и эффект sun flares – с помощью BillboardSet.

Q: ...вода?

А: Для рендеринга воды генерируется прямоугольная сетка, проецируемая на плоскость моря так, что при любом положении камеры этой сеткой будет покрываться вся часть фразума камеры, соответствующая водной поверхности. При текстурировании моря используются обыкновенные шумы Перлина для генерации затайленной карты высот, по которой строится карта нормалей для расчета отражения. Также учитывается эффект солнечной/лунной дорожки. При этом – в силу причин скорее исторического, нежели

технологического характера – рендеринг производится напрямую через DirectX. Таким образом, в данном случае сам OGRE не использовался, но не менее эффективно этот же алгоритм можно было бы реализовать и его средствами.

Q: ...взрывы?

А: Взрывы используют системы частиц из OGRE с собственными аффекторами. Здесь проявилась прекрасная расширяемость OGRE – мы смогли легко добавить новые типы аффекторов для систем частиц.

ОБЩИЕ ВОПРОСЫ

Q: Расскажите про скриптовую систему.

А: Мы использовали LuaPlus и собственную надстройку над ним, позволяющую очень легко создавать скриптовые «обертки» для объектов C++. Скрипты у нас используются довольно интенсивно: поведение игрового движка может быть в значительной степени модифицировано скриптами, в частности вся логика миссий (в том числе и обучающих) построена исключительно на скриптах. Кроме того, на



скриптах был написан редактор уровней, и с помощью набора скриптов создавались последовательности кадров для QTVR-роликов с видами юнитов для музея на официальном сайте игры.

Q: Как чистилась память?

A: Широко использовались умные указатели из нашей внутренней библиотеки с внешним подсчетом ссылки.

ОПТИМИЗАЦИЯ

Q: Насколько мне известно, использование систем частиц не отличается производительностью в ORGE. Как с этим боролись?

A: Рендеринг систем частиц был переписан, в результате чего работать они стали значительно быстрее. Кроме того, используются другие оптимизации, например системы частиц, не попадающие в видимую область, не андеятятся, что также увеличивает производительность игры.

Q: Расскажите об оптимизации в целом. Какие аспекты подверглись наибольшему переделкам и почему?

A: Самые большие изменения в OGRE как раз и связаны с оптимизацией рендеринга систем частиц. Из других значительных изменений можно упомянуть оптимизацию андеита иерархии сцены – для уменьшения количества перерасчетов трансформаций узлов сцены.

Q: Были проблемы с конкретными карточками? Какие?

A: Были проблемы, связанные с отсутствием поддержки некоторых форматов текстур в старых драйверах nVidia, но это в прошлом. Региональное представительство nVidia оказало нам помощь и поддержку



как на этапе разработки, так и на этапе тестирования, за это – огромное спасибо. А вот самым неприятным сюрпризом стали необъяснимые ошибки драйвера с последующим вылетом в «синий экран смерти» при работе на некоторых карточках Radeon 9600-9700, обнаруженные уже после выхода игры; с ними мы боремся до сих пор.

Q: Как вы в целом оцениваете степень модификации движка? В итоге это себя оправдало?

A: Безусловно, OGRE был нами значительно переработан. Но все равно мы затратили гораздо меньше сил, чем если бы разрабатывали собственный графический движок.

ЛИЦЕНЗИЯ

Q: Известно, что LGPL уже давно переведена на русский язык, а на многочисленных форумах уже даны все ответы. Но как наш, отечественный, разработчик, сделавший коммерческую игру на этом движке, пожалуйста, раз и навсегда – прокомментируйте вопросы: 1) отчисления внесшим в движок вклад разработчикам; 2) необходимость публикации исходного кода. Спасибо!

A: 1) LGPL не требует обязательных денежных отчислений авторам за использование библиотеки, распространяемой по этой лицензии, хотя если у нас будут роялти, то почему бы и нет.

2) Самым значительным отличием LGPL от GPL как раз и является то, что предоставление исходников всего проекта, использующего библиотеку с LGPL-лицензией, не является обязательным. Если не вдаваться в подробности, то в большинстве случаев при использовании динамического связывания (библиотеки в .dll) достаточно обеспечить возможность доступа к модифицированным исходникам LGPL-библиотек. Например, модифицированный нами OGRE любой желающий может скачать по адресу:

<http://pacificstorm.ru/files/Ogre-Lesta.rar>

Q: Авторы, конечно же, не беспокоили?

A: Ну, это как сказать. Создатели OGRE были восхищены проделанной нами работой. Если цитировать дословно: «We wish Lesta Studio the very best of luck with this, it's a fantastic showcase of what OGRE is capable of».

Ну что же. Использовать OGRE или нет – решать вам. За дополнительной информацией по движку обращайтесь на сайт CDR. Но очевидно одно – OGRE предоставляет разработчику значительные возможности.





ДЖЕСС ХАРЛИН

>> ЗВУКОВАЯ ФИКСАЦИЯ

КОНТРОЛЬ КАЧЕСТВА АУДИО



Динамические музыкальные темы были важной частью игровой вселенной Star Wars: Episode III Revenge Of The Sith

РАЗРАБОТКА ВЕЛИКИХ ИГР ТРЕБУЕТ КОЛЛОСАЛЬНЫХ УСИЛИЙ. ОДНАКО КОМПОЗИТОРЫ МИНИМАЛЬНО УЧАСТВУЮТ В СОЗДАНИИ ПРОЕКТА. Задача написания игровых саундтреков часто ложится на плечи программистов, тех, кто руководит разработкой игрового аудио, или тех, кто его контролирует. И все это порой в разных сочетаниях. К тому же если в разработке игры участвуют фрилансеры, то они, в отличие от штатных композиторов, как правило, считают свою роль второстепенной.

Помимо этого, тот, кто создает игровой звук, сталкивается с совершенно особым комплексом проблем. Написание музыки – наиболее неопределенная часть всей разработки. Существует неписаное правило, согласно которому оценивается игровой звук: «Есть музыка – все отлично. Нет музыки – что-то не так». Вот мнение Майкла Варда (Michael Ward), ответственного за контроль качества в LucasArts: «Звук, как ни один другой аспект видеоигр, требует чрезвычайно тесного взаимодействия тестеров и разработчиков. Если коммуникация не будет тщательно отлажена, то возникнет пропасть между тем, что хотят сделать разработчики, и тем, с чем тестерам придется столкнуться в итоге». Часто этому не уделяют должного внимания, и в результате все бремя контроля качества ложится на плечи композитора. В результате, если в команде разработчиков нет человека, полностью ответственного за качество саундтреков, композитору часто приходится контролировать треки вручную. Так как такая практика наблюдается повсеместно, неудивительно, что разработка игрового звука подразумевает утомительные репетиции и пестрит различными багами.

ЕСЛИ С ДРУГОМ ВЫШЕЛ В ПУТЬ

Однако существуют меры, которые следует предпринять, для того чтобы выкроить

время на тестирование звука и повысить качество внутренней коммуникации.

На некоторое время забудьте правило «Есть музыка...» Только контроль качества может избавить потребителя от всех багов. Тестеры каждый день, глубоко погружаясь в игру, выискивают ошибки. Если потратить некоторое время на организацию, они могут стать лучшими помощниками разработчика. Просто им необходимо указать, к чему именно стоит прислушаться. В этом случае лучше всего составить список. Во многих отделах тестирования – как у крупных издателей, так и у мелких компаний – есть сотрудники, так или иначе причастные к музыке.

Нужно обязательно выяснить, кто это именно, и добиться того, чтобы после каждого нового билда хотя бы несколько часов тратилось только на прослушивание аудиоматериалов. Это бы концентрировало внимание на поиске багов и давало больше возможностей разработчикам ввести конкретного работника в круг собственных задач. Далее необходимо объяснить тестерам, что следует делать для выявления багов.

ОШИБКИ – ОТМЕЧЕНЫ

В 2004 году компания LucasArts выпустила игру Star Wars Republic Commando. Чтобы придать игре динамичность, мы разработали сложную интерактивную музыкальную систему, которая реагировала на работу AI и скриптов, тем самым улучшая игровой процесс. По существу вся музыка сильно зависела от вещей, не находившихся под моим контролем. Незначительные изменения в окружающей обстановке, числе врагов или открытии двери иногда становились причинами таких багов, как возникновение бесконечных циклов проигрыша треков или отсутствие плавных переходов между звуковыми темами. Чтобы выявить все аудиобаги, я заручился поддержкой двух тестеров и написал им специальную справку. Через двое суток получилось то, что впоследствии я назвал «музыкальной картой».

«Музыкальная карта» была всего лишь документом Microsoft Word, в котором детально

описывались музыка в Republic Commando и указывались игровые моменты, когда должна была осуществляться смена звуковых тем. Например, одна из подобных инструкций формулировалась так: «Если играет боевая музыка, то при входе в комнату D сначала раздаются победные фанфары, а затем следует тревожный звук». Когда такие правила выявлены и подробно описаны, весь контроль качества сводится к банальной проверке всей звуковой карты и к докладу о найденных ошибках.

Несколькими месяцами позже левел-дизайнеры из компании The Collective добавляли звук к игре Star Wars: Episode III Revenge Of The Sith. И опять я провел несколько дней, составляя музыкальную карту. Потом передал ее тестерам из The Collective и своим коллегам. На этот раз не только удаленные разработчики получили инструкции, где подключать к игре звук, но и наши сотрудники контроля качества имели исчерпывающий документ, с помощью которого можно было проверить, все ли работает так, как нужно. Если музыкальная тема была вставлена не там, после того, как о багах было доложено, мы посылали левел-дизайнерам официальный запрос – и они устраняли ошибки. Хотя у меня и не было контроля над музыкальными ресурсами, но и левел-дизайнеры, и команда тестирования следовали моим указаниям. В результате получилась небольшая команда, в которой все тесно взаимодействовали друг с другом, что и позволило добиться высочайшего качества музыки.

НАЙДИТЕ ВРЕМЯ

Тестеры, специально работающие только со звуком, и создание «музыкальных карт» помогут избавиться от тех коммуникационных барьеров, которые обычно мешают разработке игрового саунда. Если вы потратите время, чтобы в борьбе с багами заручиться поддержкой других сотрудников, и составите для них общие «музыкальные карты», то композиторам не нужна будет чья-то помощь, чтобы самостоятельно добиться качественных результатов.

ДЖЕСС ХАРЛИН пишет музыку для игр с 1999 года. Сейчас он работает штатным композитором в компании LucasArts. Пишите ему по адресу jharlin@gdmag.com



КЛАРАНДА МЕРРИПЕН

«Накладных расходов не существует!»
ТОМ ПИТЕРС (TOM PETERS)

>> ДЕЛОВОЙ РАЗДЕЛ

МАКСИМУМ ПОЛЬЗЫ ОТ РУКОВОДСТВА

Чем помогут вашей студии помешанные на бизнесе

ДЕЛОВЫЕ ЛЮДИ ИМЕЮТ ОТНОШЕНИЕ К КАЖДОМУ АСПЕКТУ РАБОТЫ ВАШЕЙ СТУДИИ – ОТ УПРАВЛЕНИЯ ДЕНЕЖНЫМИ СРЕДСТВАМИ (ФИНАНСИРОВАНИЕ) ДО РУКОВОДСТВА ПЕРСОНАЛОМ (ЧЕЛОВЕЧЕСКИМИ РЕСУРСАМИ). Они формируют ваш профессиональный облик (маркетинг), работают с заказчиками (развивают бизнес) и делают возможным выход вашего продукта на рынок.

Знали вы раньше или нет, но мы, думающие только лишь о бизнесе, определяем судьбы игровых студий. Если задуматься о причинах, по которым за последние несколько лет студии покидали деловую арену, то корень зла заключается в неподобающем управлении. Оно просто убивает студию, в то время как качественное руководство только улучшает ее положение.

ЧТО ПОСЕЕШЬ, ТО И ПОЖНЕШЬ

Затраты на эффективное управление окупаются, как это и показано на следующих примерах.

Группа штатных маркетологов координировала взаимодействие издателя и студии. Их трудами процесс производства графических материалов стал настолько сбалансированным, что, когда неожиданно поступил запрос на создание трейлера для E3, был заключен контракт, который обернулся для студии дополнительной прибылью.

Одна компания наняла IT-специалиста, который помимо исполнения своих прямых обязанностей – системного администрирования – смог на сорок процентов уменьшить время компиляции. Овладев инновационной и недорогой серверной технологией, этот умелец добавил каждому разработчику по полчаса рабочего времени, которое раньше уходило на компиляцию.

А только представьте себе, чего может достигнуть любой HR-менеджер, если наладит связи с местными академическими кругами? Мне известен случай, когда один профессор и его выпускники в течение целого года проводили собственное исследование в интересах только одной студии.

ДАЕШЬ РАБОТУ ПО СПЕЦИАЛЬНОСТИ

В игровые студии требуются исключительно квалифицированные сотрудники. Мы никогда в жизни не найдем для руководства художественным отделом человека с улицы и не позволим программировать игровой движок новичку в C++. Но когда дело доходит до управленческих кадров, их обязанности часто перекладываются на плечи других сотрудников. Как часто мы видим, что программисты возятся с железом, а администраторы подбирают персонал, или, что намного страшнее, ведут финансовые дела? Таким же образом вся деловая часть зачастую возлагается на уставшего и неподготовленного руководителя студии. Маркетинг, если мысль о нем вообще кому-нибудь придет в голову, осуществляется силами художников и дизайнеров. Кто-то, обычно против собственной воли, вынужден разрабатывать веб-сайт и изредка обновлять его. Какие-то дополнительные усилия расцениваются как бесполезная трата времени.

Необходимо четкое осознание того, кто чем должен заниматься. Раз у вас замечательные программисты, дизайнеры и художники, то им нужно не менее замечательное руководство. И спрос на него не меньше.

ПЛАНИРОВАТЬ ИЛИ ДЕЙСТВОВАТЬ?

Молодые компании часто не могут себе позволить нанять сотрудника на каждую необходимую должность. Но в управлении планирование важнее, чем собственно руководство. С самого момента основания вашей студии вы должны иметь четкие представления, куда двигаться дальше в каждом конкретном направлении. Необходимо иметь тщательно проработанную стратегию, а не ломать себе всякий раз голову в поисках решений возникающих проблем.

Что касается финансовой составляющей, то в самую первую очередь следует осознать разницу между бухгалтерским учетом и управлением деньгами студии. С первого дня нужно определиться с тем,

кто будет исполнять обязанности ответственного по финансовой части, и кто будет прорабатывать возможные сценарии развития событий. Ответственно же должен заниматься бухгалтер.

Во-вторых, следует определиться с финансовым консультантом, который потребуется тогда, когда дело дойдет до получения прибыли (откуда она появится?) и поиска альтернативных источников финансирования (за счет чего вы будете финансировать ваш проект или осуществлять двухнедельные выплаты?). Не считайте само собой разумеющимся, что любой справится с этим. Вам нужен такой человек, который бы любил деньги, умел их считать и понимал, что происходит в жизни на реальном рынке.

Если вам не удастся сразу нанять специалиста, то у вас есть возможность выбора. Пойдите в экономический колледж, найдите талантливого выпускника и платите ему за помощь. Предложите известным финансистам войти в состав совета директоров и прислушивайтесь к их мнению. Пользуйтесь услугами консультантов. Читайте объявления. В них никогда нет недостатка.

ВСЕ ИЛИ НИЧЕГО?

Составить наиболее верное мнение о сотруднике, услугами которого вы собираетесь воспользоваться, можно, попытавшись ответить на весьма конкретный вопрос: «А другие компании платили ему за его услуги?» Если этот человек не блещет своими профессиональными талантами, его можно нанять как фрилансера. Если вам не удастся найти стоящего кандидата, воспользуйтесь услугами агентств по найму.

Вам нужен предприимчивый и талантливый сотрудник, на мнение которого в деловых вопросах вы могли бы полагаться. И вы, как руководитель студии, не имеете права нанимать кого-либо, не удовлетворяющего этим требованиям. Политика привлечения только самого лучшего или многообещающего персонала должна приносить плоды в каждой составляющей вашего бизнеса. Помните – то, что вы вкладываете, потом вернется.

КЛАРАНДА МЕРРИПЕН – вице-президент отдела управления компании Cryptic Studios. В этом году на GDC она постарается усмирить помешанных на бизнесе участников. Связаться с ней можно по электронному адресу cmerripen@gdmag.com



НОЙ ФАЛЬШТАЙН

>> ГЕЙМ-ШУЙ

КОГДА ОХОТНИК
СТАНОВИТСЯ ЖЕРТВОЙ

НОЙ ФАЛЬШТАЙН

(Noah Falstein) – ветеран игровой индустрии, работает в ней уже 25 лет. На его сайте www.theinspiracy.com вы найдете описание «Проекта 400», материал которого лег в основу этой колонки. Там же находится коллекция собранных на данный момент правил гейм-дизайна и советов по их использованию. Ною можно написать на адрес nfalstein@qdmag.com.

Я ВЫРОС НА СТАРЫХ ФИЛЬМАХ О ВТОРОЙ МИРОВОЙ ВОЙНЕ. Мне всегда нравились сцены, когда подводная лодка незаметно подкрадывается к вражескому конвою, транспортирующему технику или горючее. «Первая установка – залп! Вторая установка – залп! Торпеды пошли!» Затем неизменно следует: «Вражеский эсминец приближается! Срочное погружение!» Звук sireны. Потом – часы томительного ожидания.

ПРАВИЛО

Постарайтесь сделать так, чтобы охотник сам стал жертвой. Заставьте игрока сменить роль. Это поможет создать драматическое напряжение, подвигнет его проявить отвагу. Кроме того, иногда бывает просто забавно, когда опасные соперники вдруг сами становятся беззащитными. Но убедитесь в том, что наступил подходящий момент для подобного развития событий. Игрок не должен почувствовать какой-либо фальши, случайности и не должен думать, что его наказывают.

КОГДА ПРИМЕНЯТЬ
ЭТО ПРАВИЛО?

Этому правилу можно следовать во всех играх, где есть схватки или конфликт.

Когда выигрываешь от применения этого правила?

Я не смогу перечислить все ситуации, но приведу парочку примеров.

«Сделайте игру более эмоциональной, чтобы игрок почувствовал себя ее частью» (см. статью «Вопросы жизни и смерти» («A Matter of Life and Death») в декабрьском номере за 2004 год) – это общая формулировка нашего правила. Ей также созвучен тезис: «Не дайте игроку почувствовать усталость». Вы должны следовать этому совету, чтобы оправдать надежды геймеров.

ПРИМЕРЫ И КОНТРПРИМЕРЫ

Наше правило используется во многих играх. Первый и наиболее очевидный пример – это старый добрый Pac-Man. Как только игрок «съедает» энергетические бонусы, опасные приведения тотчас превращаются в легко уязвимые (и к тому же весьма ценные) цели. Начинается охота на охотника. А вот более свежий пример – Katamari Damacy. Чертовски забавно грозные враги в панике уносят ноги, когда ваш katamari увеличивается в размерах. Походные стратегии и стратегии реального времени уже давно основываются на принципе «камень / ножницы / бумага», когда один вид юнитов имеет преимущество над другим, но сам слабее третьего. Именно поэтому военные фильмы о подводных лод-

ках столь увлекательны. Часто ситуация зависит даже не от характеристик юнитов, а от того, на какой территории они находятся или в каком режиме функционируют. Это ощутимо добавляет занимательности.

Во время Второй мировой войны подводная лодка, скрывающаяся на глубине 100 метров, находилась в относительной безопасности, но не представляла угрозы для противника. С уменьшением глубины она становилась как опаснее, так и уязвимее. В игре Starcraft с успехом реализована возможность юнитов прятаться или маскироваться перед атакой, прямо как в сериале «Star Trek». В играх вообще часто используется маскировка.

Серия The Thief, да и почти весь поджанр stealth-игр, также активно использует это правило. Средневековый ассасин, затаившийся в темноте, ничем не отличается от нашей подводной лодки. Во многих RPG для того, чтобы тебя не заметили, применяется магия. Невидимый маг – мощнейшее оружие, но как только ниндзя заметит его и подберется поближе, магу несдобровать.

РЕАЛИЗАЦИЯ

Наилучший способ воплотить наше правило в жизнь – заставить игрока самого идти на риск. Это намного лучше, чем вовлекать его в авантюры. Командир подводной лодки и таинственный ассасин должны иметь возможность выждать, если их жертва слишком хорошо охраняется. Впрочем, риск может быть вполне оправдан, если он того стоит. В том случае если авантюра не удастся, игрок будет винить самого себя, а не создателей игры.

ОБРАТНАЯ СВЯЗЬ

Статья «Отрицательная обратная связь» вызвала море откликов. Так, Нейл Мескауски (Neil Meskauski) из компании Sonicube Games привел несколько примеров. Он указал на тот факт, что в игре Battlefield 2 чем больший калибр оружия используется, тем оно тяжелее и тем меньше его скорострельность. Однако он предостерег: если ООС будет реализовываться не последовательно, то игрок может почув-

ствовать разочарование и обман. Несколько читателей привели примеры из спортивных игр, за что им большое спасибо, так как я в этом жанре плохо разбираюсь. Чарльз Хаут (Charles Naut) поделился наблюдениями того, что в некоторых спортивных играх сложность меняется динамически. Если вы выигрываете, то команда соперника действует лучше, и наоборот – если вы проигрываете, то она сбавляет

обороты. Пол Терри (Paul Terry) из EA, принимавший участие в разработке большинства проектов серии High Heat Baseball от 3DO, не сомневается, что ООС часто используется в играх. Пол рассказал нам, как его команда моделировала усталость подающего. Самый лучший игрок не может все время подавать без каких-либо последствий. Дизайнеры сделали, как и в реальной жизни: игрок устает не только в

результате текущей игры, но и по ходу сезона, если выступает слишком часто. Интригуя нас, Терри добавил, что EA удалось избавиться еще от кое-каких проблем с подачей путем более точного моделирования. К тому же разработчиками часто использовался принцип «камень / ножницы / бумага». Оказывается, он часто справедлив и в реальной жизни, знаком игроку и не вызывает у него чувства обмана.



СТИВ ТЕОДОР

>> ДОЗА ПИКСЕЛЕЙ

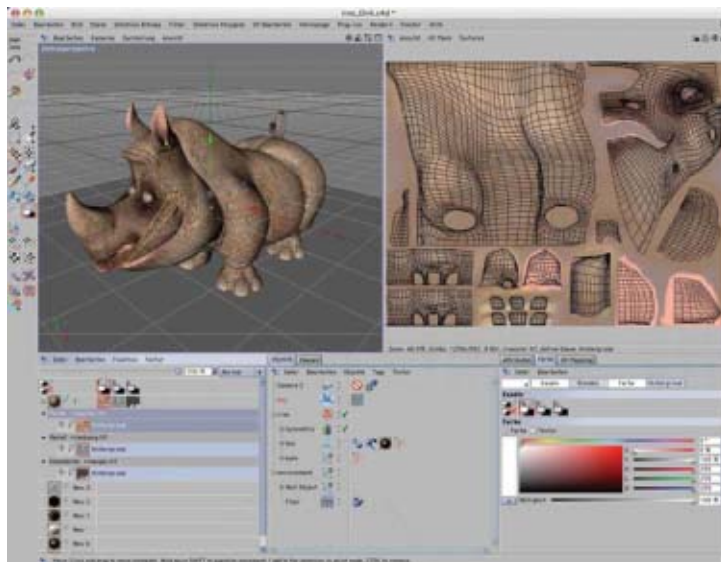
ГОНКИ НА ВЫЖИВАНИЕ

Подборка программ, которые вам стоит попробовать

РИСКУЯ РАСКРЫТЬ СВОЙ ПОЧТЕННЫЙ ВОЗРАСТ, ПРИЗНАЮСЬ ОТКРОВЕННО, ЧТО ДЛЯ МОЕГО ПЕРВОГО 3D-ГРАФИЧЕСКОГО ПАКЕТА МНЕ ТРЕБОВАЛОСЬ СОЗДАВАТЬ СЦЕНЫ В ТЕКСТОВЫХ ФАЙЛАХ. Я застал те времена, когда все приложения для моделирования и анимации можно было перечислить по пальцам одной руки.

В те далекие дни младенчества компьютерной графики каждая демонстрация от NAV или Siggraph влекла, словно витрина кондитерской лавки. Идти в ногу с прогрессом было непросто, поскольку в ранних 90-х разработчики программного обеспечения без колебаний заламывали по \$50 тыс. за полноценные анимационные пакеты. И каждый занятый в бизнесе с жадностью изучал продукты конкурентов, пичкал свои детища уникальными фишками, изобретал различия в интерфейсах и неизменно вступал в бесконечные войны платформ.

Спустя 15 лет рынок несоизмеримо вырос, но стал при этом куда менее бурным. В отличие от новичков в игровой индустрии, которые продолжают жадно изучать каждый пресс-релиз на Highend3d.com, большинству из нас давно наскучило подобное коллекционирование. Седеющие ветераны вроде меня здорово прикипели к своим основным инструментам. Нас слишком утомили производственные дедлайны, мы достаточно повидали перехваленных программ – и больше не верим в приложения, способные одним нажатием кнопки



BodyPaint 3D, модуль Cinema 4D от Maxon Software

решить любую поставленную задачу. Таковы факты: со снобизмом профессионального художника необходимо считаться. Когда весь арт на сайте производителя софта выполнен художниками-любителями (а то и вовсе самими программистами), от такого ПО не ждешь ничего хорошего.

РАЗДЕЛ «ТЕМ НЕ МЕНЕЕ»

Разумеется, после такой постановки вопроса проникательный читатель ожидает услышать «...тем не менее...». Что ж, вот вам, пожалуйста.

Тем не менее наша трудовая жизнь хитро переплетена с продолжающимся развитием 3D-приложений, и постоянно следить за обновлением ПО важно для нашей профессии по трем причинам.

Во-первых, вновь почувствовать энтузиазм, который прежде заставлял нас ночами просиживать за UV-текстурированием вместо

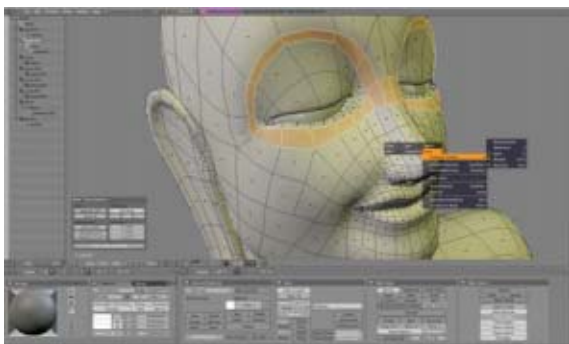
того, чтобы изобретать со сверстниками новые коктейли, может быть, с одной стороны, просто забавно, а с другой – еще и крайне полезно для карьеры. Отслеживание появления новых программ должно стать одним из ваших обязательных навыков. Рассматривайте это как профессиональную самооборону.

В качестве конкретного примера приведу молниеносный взлет Zbrush, который из неприметного любительского инструмента всего за несколько лет дорос до законодателя мод в своей отрасли. В целом индустрия только выиграла от такого вливания энергии и новых приемов.

Как бы то ни было, позиции тех, кто в своей карьере сделал ставку на достаточно специфические технические умения (вроде моделирования убедительной головы при помощи би-сплайнов), довольно серьезно пошатнулись после того, как оперативное иерархическое моделирование стало достоянием масс.

СТИВ ТЕОДОР начинал анимировать еще в текстовом интерфейсе мэйнфрейма, а впоследствии работал над такими играми, как *Half-Life* и *Counter-Strike*. Для связи с ним пишите на stheodore@gdmag.com

ДОЗА ПИКСЕЛЕЙ



Blender с открытым кодом – альтернатива многим 3D-графическим пакетам

Естественно, лучшие художники выигрывают в конце концов – но тем, кто идет не в ногу со временем, до высокого результата топать и топтать. Так что если чисто интеллектуальное любопытство не побуждает вас скачивать новые демо-версии, то просвещенное своекорыстие должно на вас повлиять.

Второй повод неделями играть с новыми программами в забавную игру «Отыщи-ка все горячие клавиши!» состоит в том, что нет ничего показательнее для оценки сильных и слабых сторон привычного вам рабочего процесса, нежели непродолжительная смена условий труда. Легко смириться с тем, что сложности, с которыми вы сталкиваетесь в привычных условиях, – это непреложные издержки производства. А вдруг препятствия, которые вам ежедневно приходится преодолевать, вызваны вторых прикрученной фичей, недопонятой программной спецификацией или банальной некомпетентностью поставщика ПО?

И наконец, увидеть полноценную реализацию приема, который до этого встречался вам только в бестолковом воплощении, дорого стоит. Это может рассказать вам куда больше о механике работы вашего графического пакета, нежели часы терпеливых разъяснений программистов (если предположить, что терпеливые программисты существуют). К тому же, если вы пользуетесь тем, о чем прочие художники едва знают, вы можете заработать репутацию на

эксклюзивности. И хотя мы все в Game Developer твердо уверены, что художники, которые держат свои секреты при себе, вредны для индустрии, трудно отрицать, что припрятанный в рукаве туз подчас может здорово прибавить очков вашему рейтингу.

ЗОЛОТОЙ ВЕК ДЕМО-ВЕРСИЙ

К счастью для нас, сейчас самое подходящее время для художников, которым не чужды эксперименты. Распространение 3D-графического оборудования и соответствующих стандартов значительно упростило создание трехмерных приложений. Приемы, которые в свое время являлись чуть ли не таинствами, знакомы теперь многим программистам. К примеру, инструмент для иерархического моделирования сегодня создать проще, чем когда-либо прежде. Помимо всего прочего, рынок графических приложений вырос неимоверно: к традиционным кино- и телекомпаниям добавились игровые студии, веб-дизайнеры и любители, возмнившие себя профессионалами. В результате даже небольшие конторы сегодня производят впечатляющие продукты.

Еще один повод считать, что мы живем в золотой век экспериментаторства по созданию инструментария для художников, – это доступность демо-версий. Во времена, когда анимационный пакет стоил больше, чем новенькая BMW, получить копию софта было целым предприятием. Однако с выходом Maya Personal Learning Edition и Gmax производители начали понимать, что создание пользовательских кадров – основной способ обеспечения будущих продаж. Добавьте сюда доступность широкополосных каналов – и вы можете самостоятельно опробовать любой значимый продукт в отрасли ценой места на жестком диске и нескольких часов ожидания, пока завершится загрузка.

С учетом всего этого мне бы и хотелось выделить несколько демо-версий, ознакомиться с которыми я бы рекомендовал. Уч-

тите, однако, что информация ниже вовсе не претендует на статус обзора. Обзоры призваны помочь в выборе покупки – наши же сведения всего лишь укажут вам на продукты, которые будет интересно и полезно попробовать. Кстати, если вы не встретите в данном перечне продукта, который считаете достойным, напишите мне – и я упомяну его в следующей статье.

ВЕДРО ПРОГРАММ НА \$15 ТЫСЯЧ – БЕСПЛАТНО!

Изобразительные инструменты сейчас так стремительно развиваются, что в одном материале я никак не сумел бы упомянуть все демо-версии. Как бы то ни было, представляю список продуктов полного цикла – объединенных программных пакетов для моделирования, анимации и рендеринга, которые призваны стать локомотивом любого профильного производства. И хотя в игровой индустрии царят Max и Maya, найдется еще с полдюжины схожих по набору функций, столь же технологически продвинутых и – что немаловажно – бесплатных демо-версий. Даже если вы много лет работали в Max и Maya и ни разу им не изменяли, вам все равно не помешает скачать обучающую демо-версию и посмотреть, как работают все остальные.

Итак, я прекращаю дозволенные речи: перед вами в алфавитном порядке представлены восемь демо-версий, которые рекомендуется попробовать.

3DS MAX И GMAX, AUTODESK

www.autodesk.com

На странице Autodesk вам предложат скачать полноценную пробную версию 3DS Max 8, которая ограничена по времени 30 сутками. Демо-версия идентична лицензионной копии на весь ознакомительный срок. До недавнего времени на сайте Autodesk также предлагался Gmax – бесплатная, но урезанная в фичах версия Max,

предназначенная для моделирования и конструирования уровней. Gmax пропал с Autodesk в октябре прошлого года, но вы, вероятно, сумеете отыскать копию, если поищете: этот продукт очень популярен среди создателей модов.

Только под Windows2000/XP.

BLENDER, BLENDER FOUNDATION

www.blender.org

Вообще говоря, у Blender'a нет демо-версии. Дело в том, что она ему и не нужна: это бесплатный продукт с открытым кодом. Хотя открытый код зачастую ассоциируют с дилетантством, Blender очень функционален и оригинален. Также он предлагает специальную поддержку для разработчиков игр: интегрированный язык скриптов Python, встроенная GL-графика и внутриоконные интерфейсные слои. Самую новую, полную, без каких-либо ограничений версию (как и ее полный исходный код) можно свободно скачать с Blender.org и использовать по лицензии GNU. Если Blender вам пригодился, не поскупились на денежное пожертвование или впишите Blender в титры.

Как и положено продукту с открытым кодом, Blender работает под Windows, Mac OS X, Linux и Solaris.

CINEMA 4D 9.5, MAXON SOFTWARE

www.maxon.net

Cinema 4D – это достаточно популярный в Европе модульный пакет для моделирования, рендеринга и создания анимации. Он щеголяет гибко настраиваемым интерфейсом, изощренным алгоритмом рендеринга и встроенным языком скриптов. Один из особо интересных модулей – BodyPaint 2.5, встроенный инструмент для рисования в 3D. Укомплектованная всеми фишками ознакомительная версия Cinema 4D практически идентична полному продукту, за исключением возможности сохранения, и доступна на сайте Maxon в версиях под Windows и Mac OS.

HOUDINI APPRENTICE, SIDEFX SOFTWARE

www.sidefx.com

Houdini от Sidefx.com имеет глубокие корни в кино- и телебизнесе, но никогда прежде не заявлял о себе в производстве игр. Продукт сочетает привычные инструменты для моделирования, анимации и рендеринга с очень сильной процедурной ориентацией – почти каждой задачей по моделированию, настройке или анимации модели можно управлять при помощи цепи регуляторов, выстроенных в некий аналог блок-схемы. Еще один плюс – это адекватная встроенная система компоновки изображения: ключевая особенность, неизменно привлекательная для кино- и телерынка.

Houdini Apprentice выдает изображение с разрешением до 440x480 пикселей с водяным знаком и в защищенном формате. Дистрибутив под Windows и Linux, а также сопутствующая документация доступны на сайте SideFX.

LIGHTWAVE DISCOVERY EDITION, NEWTEK

www.newtek.com

За LightWave долго держалась репутация производителя оригинальных продуктов с явственно различимыми особенностями. Хотя завсегда Max и Maya модульная организация работы в LightWave покажется непривычной, он представляет собой ценную перспективу развития графических пакетов полного цикла, а также несет множество интересных интерфейсных находок. Newtek предоставляет бесплатный образец продукта – Discovery Edition – с ограничением на количество вершин в моделях и водяными знаками на изображении. К несчастью, отыскать место, где можно скачать данную версию, не так-то просто. Впрочем, вы просто можете приобрести менее чем за \$50 одну из обучающих книг к программе: на прилагаемом диске непременно будет демо-версия.

MAYA PERSONAL LEARNING EDITION, ALIAS

www.alias.com

Maya's Personal Learning Edition – это ограниченная версия Maya Complete. Изображение, отрендеренное при помощи данной версии, ограничено по разрешению до 1024x768 пикселей и снабжено водяным знаком. В Learning Edition доступны все базовые инструменты Maya, однако запрещено использование плагинов и данных о модели. Также нет доступа и к языку скриптов MEL. Данная версия умеет открывать файлы, сохраненные в Maya Complete, однако сама записывает данные в закодированный .trp-формат, который другие версии Maya прочесть не могут. Personal Learning Edition работает под Windows 2000/XP и Mac OS X.

SOFTIMAGE XSI MOD TOOL, SOFTIMAGE

www.softimage.com

В те дни, когда графическая рабочая станция обходилась в \$40 тыс., Softimage был одним из широко используемых программных пакетов, в котором появилась одна из самых ранних коммерческих реализаций инверсной кинематики. В 2000 году умельцы из Монреаля возжаждали хотя бы толики былой славы и выпустили Softimage XSI. В текущей 5-й версии XSI обладает достойным набором инструментов и очень современным интерфейсом.

Демо-версия Softimage называется Mod Tool. С ее помощью можно создавать изображения разрешением до 510x510 пикселей, помеченные водяным знаком. Модели демо-версия сохраняет (по примеру Maya Personal Learning Edition) в защищенные файлы и помимо этого несет еще ряд защитных механизмов, не допускающих полноценного использования программы. В Mod Tool есть плагин, сохраняющий модели для Unreal Engine от Epic и Source Engine от Valve. Эту фику Mod Tool позаимствовал из Softimage 4.2 (текущая версия продукта – 5.0). Mod Tool можно скачать с сайта Softimage в версии под Windows 2000/XP.



Houdini Apprentice позволяет художникам бесплатно экспериментировать с продуктом от SideFX



МИК ВЕСТ

>> ВНУТРЕННИЙ ПРОДУКТ

МНОГОЯДЕРНЫЕ ПРОЦЕССОРЫ

ДЛЯ УВЕЛИЧЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ ИГРОВЫЕ ПЛАТФОРМЫ НОВОГО ПОКОЛЕНИЯ ВСЕ ИНТЕНСИВНЕЕ ИСПОЛЬЗУЮТ МОЩЬ НЕСКОЛЬКИХ ПРОЦЕССОРОВ. Внутри Xbox 360 находятся три симметричных CPU, каждый из которых одновременно обрабатывает два потока. У PlayStation 3 один главный CPU и семь дополнительных «SPE» процессоров. Что касается Nintendo Revolution, то технические спецификации по этой платформе еще не были обнародованы, но я предполагаю, что консоль будет двухядерной. Современные компьютеры, предназначенные для игр, — двухядерные. Программисты, до последнего времени писавшие код для одноядерных процессоров, сейчас сталкиваются с проблемой эффективного использования собственных навыков в многопроцессорных системах.

В идеале хотелось бы использовать всю вычислительную мощность всех доступных CPU. Весьма прискорбно, когда процессор простаивает.

В своей статье я коснусь вопросов, связанных с программированием для многопроцессорных машин, и опишу некоторые подходы, следуя которым разработчик сможет так спроектировать свой игровой движок, чтобы нагрузить процессор по полной.

НЕ СТОИТ?

Самый легкий выход — это попросту проигнорировать дополнительный процессор и выполнить игровой код как один поток на отдельном CPU (см. рисунок 1).

Договоримся, что под «визуализацией» имеется в виду суммарный вклад процессора в высокоуровневый процесс определения видимости объектов и подготовку передачи потоков данных на GPU. Обработка данных на GPU не показана.

В данном случае процесс выполнения программы целиком и полностью линейен. За выполнением игровой физики следует игровая логика, и, наконец, CPU вносит свой вклад в визуализацию. Каждая из этих высокоуровневых задач разбивается на ряд последовательных подзадач промежуточного уровня, выполняемых в строго определенном порядке.

Так, например, физическая подсистема сначала обрабатывает твердые тела, затем инверсную кинематику и затем системы частиц. Конечно, в реальном игровом движке все может происходить гораздо сложнее. Очевидно, что хотя однопоточный код не использует возможных преимуществ многоядерного CPU, но зато программирование игры весьма упрощается.

Целесообразность подобного подхода зависит от многих факторов — от типа разрабатываемой игры, сроков и доступных вам ресурсов.

Помните, что за качество выдаваемой картинки все еще отвечает GPU. Если в игре относительно немного интерактивных объектов и анимации (например в симуляторе гольфа), можно избежать сложностей, используя только один процессор. Дополнительным плюсом этого решения будет значительное упрощение процесса отладки. Многопоточные приложения очень сложно отлаживать.

Если у вас мало времени и уже есть код работающего однопоточного кода, с точки зрения возможных издержек может быть эффективнее проигнорировать дополнительный CPU — по крайней мере для данной версии игры.



Рисунок 1.

Одно ядро бездействует, один поток

УПРАВЛЕНИЕ ПОТОКАМИ ВРУЧНУЮ

Многие задачи промежуточного уровня довольно независимы друг от друга, так что они могут исполняться в любом порядке, даже параллельно. Если у вас есть такие задачи, то и выполняйте их параллельно — сэкономите время, так как общее время выполнения будет равно времени выполнения более сложной задачи.

Рисунок 2 иллюстрирует ситуацию, когда частично задействовано второе ядро. Физика твердых тел и ИК-анимации теперь обрабатываются одновременно.

Работа игровой логики не разделена на потоки. Обычно код, ответственный за AI игры, сильно зависит от очередности выполнения программы. Очень сложно разделить работу игрового AI на несколько потоков так, чтобы потом не получить массу хитроумных багов.

МИК ВЕСТ является со-основателем Neversoft Entertainment. Он в индустрии около 17 лет, в настоящее время работает техническим консультантом. Пишите ему на mwest@gdmag.com

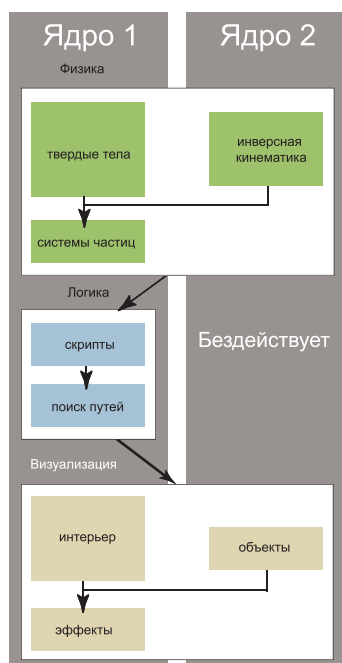


Рисунок 2. Некоторые подсистемы выполняются на втором ядре

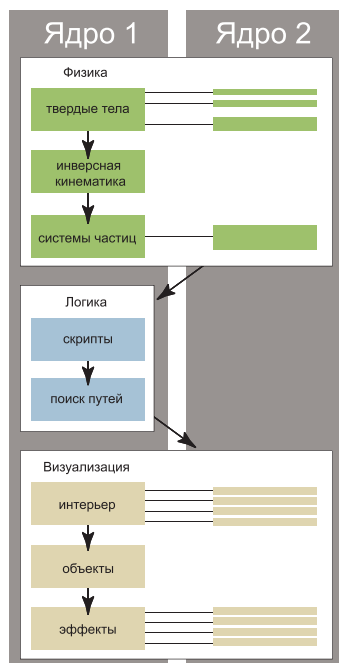


Рисунок 3. Все системы распараллелены на уровне объектов. Очередность обработки фиксирована

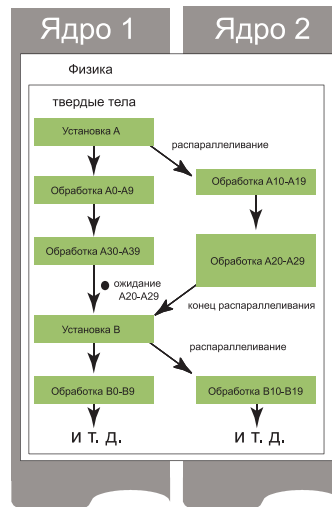


Рисунок 4. Каждый распараллеленный поток одновременно обрабатывает несколько объектов. Очередность обработки может меняться

Процесс визуализации также можно частично распараллелить (это показано на рисунке 2). Тогда интерьер визуализируется вместе с находящимися в нем объектами.

Подобный подход – когда вы вручную разделяете выполнение больших задач на отдельные потоки, которые потом выполняются разными ядрами процессора, – можно с успехом применять. Это быстро, легко и не потребует модификации игрового движка.

Реализация такого подхода может меняться в зависимости от связей подсистем вашего игрового движка. Например, невозможно одновременно визуализировать интерьер и объекты в том случае, если они используют какие-либо общие ресурсы, например потокобезопасный вершинный буфер.

Недостаток этого метода в том, что обычно немного задач выполняется параллельно. К тому же очень редко можно обойтись без модификации движка.

Так как выполняемые параллельно задачи обычно делают различные вещи, как правило, неоптимально использовать общий кэш.

Подводя итоги, можно сказать, что дополнительные ядра процессора будут использоваться весьма незначительно.

РАСПАРАЛЛЕЛИВАНИЕ ПОТОКА

Если высокоуровневые подсистемы вашей игры невозможно одновременно запускать на выполнение, вам следует обратиться к низкоуровневым. Очень часто небольшой набор команд обрабатывает значительный объем данных. Если порядок обработки объектов не играет роли, можно использовать небольшой по времени функционирования поток, работающий на другом ядре.

Его можно изобразить так: основной поток разделен на два или более, каждый обрабатывает одни и те же данные, при этом пропуская обработанные другими потоками до тех пор, пока все данные не будут обработаны.

На рисунке 3 показан порядок выполнения. Главный поток на ядре 1 идентичен изображенному на рисунке 1. Единственная разница заключается в том, что на определенном этапе этот поток разделяется на несколько мелких.

Возможность управления порядком выполнения команд является большим плюсом, поскольку позволяет значительно упростить модификацию движка под многоядерный процессор.

К тому же увеличивается эффективность использования кэша, потому что оба

потока обрабатывают расположенные почти в одном и том же месте данные.

Рисунок 4 иллюстрирует, каким образом происходит распараллеливание обработки объектов. Мы одновременно обрабатываем по 10 объектов. Сначала осуществляем подготовительный этап на наборе объектов А. Затем оба ядра одновременно начинают обрабатывать объекты, причем ядро 1 – А0-А9, а ядро 2 – А10-А19. В этом примере время выполнения потоков различно.

Как только ядро 2 завершает свою задачу, оно принимается за объекты А20-А29, а чуть позже ядро 1 – за А30-А39. Очевидно, что порядок выполнения обработки объектов не может быть заранее предсказан. Он может меняться от кадра к кадру и от запуска к запуску. Ситуация, когда порядок обработки объектов не может быть строго фиксирован, может привести к чрезвычайно трудно выявляемым ошибкам.

Однако если удастся избежать подобных трудностей, подобный подход предоставляет очень широкие возможности. Особенно приятно, что чем больше процессорных ядер в вашей системе, тем проще этих трудностей избежать.

ВНУТРЕННИЙ ПРОДУКТ

Рисунок 5. Физическая подсистема работает одновременно с логической и подсистемой визуализации. Второе ядро задействовано почти на 100%

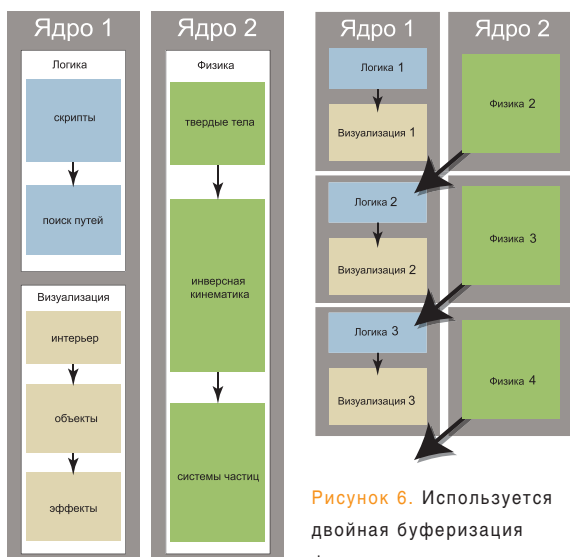


Рисунок 6. Используется двойная буферизация физических данных

НЕТ ЗАКОНУ АМДАЛА!

В соответствии с «законом Амдала» все вышеупомянутые методы имеют общий недостаток. Амдал доказал, что большая часть программного кода будет исполняться последовательно, и этого никак не избежать. Таким образом, независимо от того, сколько процессоров и ядер в нашей системе, мы получаем серьезный ограничивающий фактор.

В некоторой степени этого можно избежать, разделив ваш программный код на несколько потоков и выполняя их параллельно. В таком случае преимущества многоядерности будут использоваться наиболее полно.

На рисунке 5 (стр. 32) показано, как подобный подход выглядит во время обработки одного кадра. Оба ядра используются на 100 процентов – первое отвечает за игровую логику и визуализацию, второе обрабатывает физику.

Если игровая логика и визуализация будут осуществляться до того, как их обрабатывает физическая подсистема, наверняка придется столкнуться с трудностями.

Обойти это можно, используя полученные

данные о физике из прошлого кадра, предварительно создав их копию.

Рисунок 6 иллюстрирует работу этого метода на примере трех кадров. Его суть проще понять, если начать со второго кадра. Игровая логика и визуализация работают на ядре 1, используя данные, полученные во время обработки предыдущего кадра. В то же самое время ядро 2 обрабатывает физику, а полученные данные будут использоваться игровой логикой в третьем кадре.

Подобная архитектура должна быть хорошо знакома тем, кто сталкивался с особенностями визуализации на PlayStation 2. Там GPU – это отдельный мощный процессор, как правило, работающий параллельно с CPU и использующий физические данные прошлого кадра. Таким образом происходит расширение конвейера.

Теоретически возможно (если, конечно, доступны дополнительные ядра) распараллеливать любые подсистемы движка. Но с этим связано множество проблем. Во-первых, зачастую разные подсистемы работают с разной скоростью, так что иногда процессор будет простаивать.

К тому же иногда необходимо напрямую управлять работой физической подсистемы. Потенциально куда более серьезная проблема заключается в том, что в процессе игры могут возникнуть лаги. За время обработки пользовательского ввода может пройти несколько кадров. В случае однопоточного приложения эта цифра в среднем составляет 2.5. Обычно это приемлемо, но если в ваш конвейер будет добавлен еще один этап, то цифра вырастет до 3.5. Если же игра выдает 30 fps, возможная задержка может составить 133 миллисекунды, что допустимо разве что в Интернете. В случае 60 fps задержка будет 66 миллисекунды.

PPU и GPU

PPU – это физический процессор, предназначенный для эффективных вычислений, используемых при физическом моделировании. Он может выполнять функции ядра 2

(см. рисунки 5 и 6), будучи при этом гораздо производительнее, чем отдельное ядро CPU.

Современные GPU стали настолько мощными, что на них даже можно производить некоторую часть физических вычислений. Основная трудность в том, что они разрабатывались для функционирования в однопоточном конвейере. На практике неэффективно передавать полученные GPU результаты в начало конвейера. Однако вы можете отделить от общего моделирования физики те эффекты, которые носят косметический характер и не влияют на большую часть игрового мира. Разумеется, это будет неверно в случае системы частиц, взрывов и обломков предметов – тогда пострадает убедительность.

GPU может взять обработку физики (или ее часть) на себя и производить вычисления непосредственно на стадии визуализации. Таким образом, значительно снижаются требования к пропускной способности между GPU и PPU.

ВЫБИРАЕМ...

Смешанный подход может дать очень хорошие результаты. Конечно, существуют куда более сложные методы эффективного использования многоядерных процессоров. Но даже просто комбинируя отдельный физический «конвейер» (рисунок 5) и распараллеливание потоков (рисунок 3), вы можете без дополнительных сложностей подстраиваться под различные архитектуры. Распараллеливание работы физической подсистемы позволит вам со временем использовать преимущества отдельного PPU.

ССЫЛКИ

Wu, David, «Physics in Parallel: Simulation on 7th Generation Hardware»
www.cmp.events.com/Sessions/GD/PhysicsinParallel.ppt

Intel Corporation, «Threading Methodology: Principles and Practices»
<http://tinyurl.com/atsb5>

Gabb, Henry and Adam Lake, «Threaded 3D Game Engine Basics», Gamasutra.com
www.gamasutra.com/features/20051117/gabb_01.shtml

>> РИЧАРД ЛЕМАРЧАНД



ОТ ЗАДУМКИ К ПОБЕДЕ

Jak X: Combat Racing. Метод производства The Naughty Dog

ОСЕНЬЮ 2004 ГОДА МЫ В СТУДИИ NAUGHTY DOG INC (НАМИ СОЗДАНЫ СЕРИИ ИГР BANDICOOT И ЖАК) РЕШИЛИ СДЕЛАТЬ ИГРУ В ОТНОСИТЕЛЬНО НОВОМ ДЛЯ СЕБЯ ЖАНРЕ. На все про все у нас было всего десять месяцев. За всю историю нашей серии Jak самый короткий срок, за который нам удалось сделать игру, также составлял десять месяцев. Мой рассказ будет о том, как Naughty Dog смогла, несмотря на все трудности, справиться с задачей. Нам не пришлось идти на компромиссы в вопросах качества и на какие-либо жертвы среди сотрудников.

ДОБРО ПОЖАЛОВАТЬ НА СТАРТ!

Для того чтобы работать настолько эффективно, насколько это только возможно, перед началом каждого нового проекта мы

составляем простой и четкий план того, что нам предстоит сделать. Сотрудничая с продюсерами из Sony, один из руководителей нашей студии и всего игрового проекта Иван Веллс (Evan Wells) определил три основные цели игры.

Во-первых, мы собирались делать боевую гоночную игру. Предполагалось использовать физический движок, на котором основывался геймплей Jak 3, поскольку он получил множество положительных отзывов. Во-вторых, это должна была быть первая многопользовательская онлайн-игра нашей студии. Мы сильно волновались и очень долго предвкушали это событие. И, наконец, предполагалось создать весьма проработанный игровой сюжет и примерно сорок минут анимационных роликов кинематографического качества, которыми так славилась наша серия Jak.

РИЧАРД ЛЕМАРЧАНД (Richard Lemarchand) – руководитель разработки игр студии Naughty Dog. Он был главным геймдизайнером игры Jak X: Combat Racing. Кроме того, он работал над тайтлами Jak 3, Soul Reaver, и Gex. Написать ему письмо можно по адресу rlmarchand@qdmag.com



ОТ ЗАДУМКИ К ПОБЕДЕ

Jak X: Combat Racing. Метод производства The Naughty Dog



Скетчи персонажей к игре Jak X: Combat Racing создавал Боб Рафей (Bob Rafei)

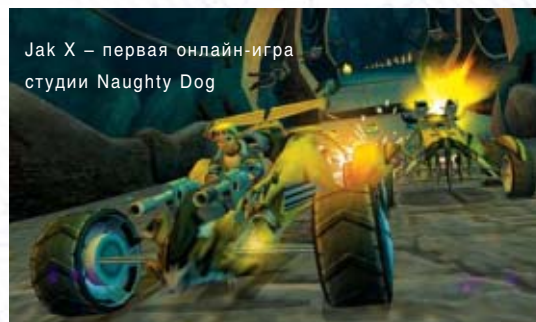
Чтобы работать по-настоящему быстро, нужен хороший план, которому необходимо scrupulously следовать. Кроме того, нельзя позволять себе делать паузы и попусту тратить ценные ресурсы.

МЕНЯЙ ПРИВОД!

Любой новый тип игры требует выработки новых методов и новых подходов к проектированию. Однако мы применяли довольно много и того, и другого так же, как и в Jak 3. Наши подходы к разработке весьма технологичны, мы никогда осознанно не следуем каким-либо догмам. Наша философия «быстрой разработки» определяет большую часть наших методов.

Например, при реализации всех наших игровых проектов мы стараемся создавать код таким образом, чтобы он по возможности в любой момент представлял собой функционирующую программу. У нас никогда не было такого, чтобы игра не работала или работала некорректно дольше, чем один день. Наша концентрация на сроках сдачи и частый выпуск демонстрационных версий позволяет быть более сосредоточенными и почти не испытывать особых стрессов. Иногда, правда, приходится хорошенько понервничать, когда нужно выполнить недельный план или выпустить новый диск к E3 или очередному пресс-релизу.

Мы осуществляем разработку «концентрически». Что это значит? Сначала мы доводим до блеска базовую механику игры. Затем приступаем к реализации дополнительных возможностей движка и не слишком существенных аспектов. Подобный взвешенный подход к проектированию, когда отсутствуют явные зависимости между элементами движка, позволяет избежать дальнейших проблем при его доработке. Такая стратегия дает нам возможность довольно быстро получить отлично работающий каркас игры и тотчас же указывает нам дальнейшее направление его модификации. Если что-то получается не столь увлекательным, как предполагалось (подобное случилось с некоторыми прототипами вооружения), это можно было безболезненно вырезать, не оглядываясь на возможные последствия.



Jak X – первая онлайн-игра студии Naughty Dog

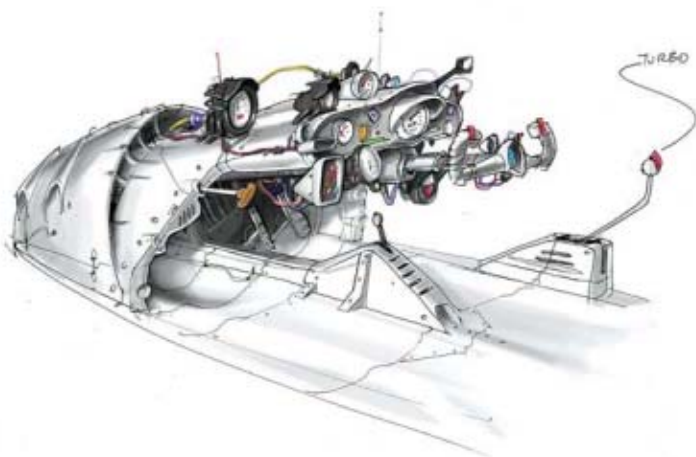
Характерная черта «быстрой разработки» – приоритет в написании работающего кода, а не длинных спецификаций. Это позволит вам кое-что изменить в своей жизни и преодолеть всевозможные кризисы. Мы считаем, что «быстрая разработка» куда лучше подходит для трудолюбивой и квалифицированной команды, нежели метод постепенного создания компонентов игры. К тому же она делает потенциально напряженную разработку куда более веселой и занимательной. Делать игру становится чуть менее увлекательно, чем играть в нее.

Другой метод, используемый нами как при создании Combat Racing, так и в двух наших предыдущих проектах, состоит в том, что мы адаптировали движок от предыдущей игры. Это позволило нам сэкономить огромное количество времени, которое впоследствии было потрачено на разнообразные мелкие улучшения.

Это был наш первый онлайн-проект, и нам пришлось впервые разрабатывать ядро такой игры. Обычно мы начинаем разработку однопользовательского проекта с создания пары-тройки миссий. Уже готовая рабочая версия дала нам возможность сделать определенные выводы, позволившие лучше осуществлять дальнейшее планирование. Во-первых, она дала нам возможность окончательно убедиться в том, что наша игра действительно увлекательна. Во-вторых, уже на раннем этапе мы осознали необходимость введения специальных средств, которые позволяли бы игрокам определять свое географическое положение в игре.

Другой метод, позволивший нам быстро двигаться вперед, состоит в том, чтобы начиная с самых первых дней разработки делать еженедельный билд игры. Это отлично вписывалось в нашу философию – «пусть ваша игра всегда будет работающей» – и принесло немало дивидендов. Всего лишь спустя пару месяцев мы, уже будучи у себя дома, каждую пятницу играли в сети друг против друга. И потом некуда было деться от разговоров о том, кто проиграл или выиграл.

Хотя готовый игровой движок у нас уже был, пришлось немало потрудиться, чтобы он полностью отвечал новому стилю. Ведущий программист Naughty Dog Кристоф Балеста (Christophe Balestra) на протяжении всего проекта направлял усилия в нужное русло: когда необходимо было осуществить весьма сложную сетевую реализацию некоторых объектов, вносить насущные изменения, чтобы отвечать новому игровому стилю, и дополнительно модифицировать физический движок.



Весьма существенным было и то, что каждый по мере создания игры мог попробовать самую последнюю ее версию. Это было возможно благодаря тому, что мы с самого начала создали специальный механизм, позволявший легко осуществлять доступ к игровым данным.

РАЗОГНАЛИСЬ!

Нам было известно, что при столь коротком цикле разработки придется выложиться до конца, и мы это сделали. Примерно за пять месяцев до официального старта проекта, который состоялся в ноябре 2004 года, один из наших ведущих программистов Бен Страгнел (Ben Stragnell), ветеран индустрии и эксперт в программировании сетевых приложений, начал писать код, отвечающий за сетевую поддержку. Впоследствии этот код был интегрирован в базовый движок Jak X.

К тому же мы заранее укрепляли позиции на фронте геймдизайна. О том, как будет сделан Jak X, мы начали думать даже еще до того, как Jak 3 пошел в тираж. Мы были уверены, что исчерпывающим образом зафиксировали идеи, первоначально пришедшие нам в голову. В дальнейшем нам представился случай убедиться, что ранние соображения по поводу геймплея были одними из самых удачных.

Наши приготовления были завершены в тот момент, когда мы скопировали данные и код Jak 3, быстренько его протестировали и откомпилировали. У нас в руках теперь была конкретная модель, с которой можно было проводить эксперименты. Но самое главное, в нее можно было играть.

ДИЗАЙН ПРИДУМАЛИ ЗАРАНЕЕ

Небольшой коллектив наших геймдизайнеров в течение первых трех недель собирался почти каждый день на два или три часа для мозгового штурма, результаты которого тотчас фиксировались. Заключительным итогом этих сборов стал весьма лаконичный дизайн-документ, который занимал тридцать страниц.

Дизайнеры обсуждали три главные темы: основные цели каждого уровня, основу сюжетной линии и основные игровые «фишки». Последние включали в себя вопросы технического оснащения машин и общей экономической структуры игры. Помимо этого на нескольких страницах были зафиксированы основные идеи игровых локаций, важных ситуаций и оружия. В конце концов туда же последовали все идеи, от которых мы отказались, равно как и некоторые дополнительные мысли. Мы намеревались проектировать весь геймплей на основе отдельных составляющих, таких как локации, события и так далее. Планировалось использовать специальные комбинации составляющих, при которых игра приносила бы наибольшее удовольствие.

Дизайн-документ был представлен команде, и впоследствии его модифицировали несколько раз еще на первоначальном этапе разработки. В оставшееся время нами использовалась система «wiki», которая обеспечивала реализацию гибкой системы документации и осуществляла привязку идей к срокам их реализации. Да и вообще,

эта система предоставляла всей команде документацию, которая либо могла быть полезной, либо попросту отсутствовать.

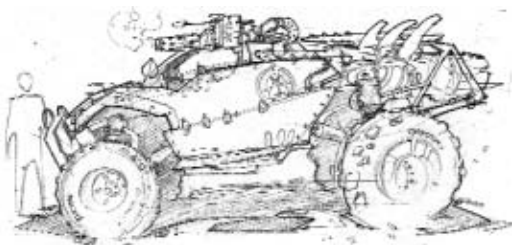
Задолго до этого проекта мы обсуждали плюсы и минусы геймплея, основанного на физике. Нами были предсказаны некоторые изменения, которые следовало бы внести для достижения баланса в игре. Прогнозы оказались верны. Они позволили нам заранее подготовиться к трудностям и запланировать соответствующее время на их устранение.

Благодаря Дэну Эри (Dan Arey), сценаристу и креативному директору Naughty Dog, наша сюжетная линия стала куда более нелинейной. Развитие сюжетной линии зависело от определенных факторов, которые легко можно было изменить. Последнее чертовски нам пригодилось, когда мы несколько раз его переделывали. Наша прекрасная команда аниматоров приложила огромные усилия, чтобы в срок выполнить свои задачи. Они, как и раньше, добились блестящих результатов. Особо стоит отметить тот факт, что вводные ролики были самыми сложными из тех, которые вообще создавались в нашей студии.

Другая важная дизайнерская работа состояла в том, что мы с самого начала представляли себе игровой процесс и интерфейс. Геймплей в прошлых играх серии был довольно простой. Однако так как разрабатывалась именно онлайн-игра, мы должны были привлечь внимание игрока. Нам необходимо было заставить его все время думать о характеристиках своего героя и его машины. С самого начала нам был просто необходим серьезный предварительный план.

После тщательного анализа некоторых проектов конкурентов мы составили всеобъемлющий список «фич», которые, как нам кажется, позволят делать игроку то, что ему только заблагорассудится. Сказать по правде, с самого начала нам не удалось прийти к окончательному варианту, так что этот список мы несколько раз переделывали.

Мы также потратили много сил на то, чтобы создать предварительный вариант интерфейса и провели большую часть ноября, тестируя интерфейс на предмет его функциональных возможностей, которые нас постоянно не устраивали. Интерфейс постоянно улучшался, и в конечном счете с помощью технологии Flash был создан его прототип. Мы решили использовать наши возможности в трехмерном моделировании на полную катушку, и вместо изначально планируемого двумерного интерфейса сделали стильный и анимированный полигональный интерфейс. Это заняло уйму времени, но мы были просто в восторге от результатов.





ОТ ЗАДУМКИ К ПОБЕДЕ

Jak X: Combat Racing. Метод производства The Naughty Dog



Уникальный внешний облик игровых персонажей выделяет серию Jak на фоне остальных игр

Наконец, художественный директор Боб Рафей (человек, создавший стиль серии Jak и Crash Bandicoot) вместе с другими замечательными художниками начал работать над образами, которые должны были определить правильное направление работы. Создание бумажных эскизов поможет сэкономить уйму времени (и денег) и вдохнуть новую жизнь в дизайн игры. Мы убедились, что наконец-то выбрали нужное направление, когда закончили наши замечательные концепты машин (выражаем благодарность Хуго Мартину (Hugo Martin), талантливому проектировщику автомобилей), интерфейса и... баннеров для трасс.

Принимая во внимание те временные затраты, которые мы понесли, можно с уверенностью сказать, что планирование критически важно для того, чтобы ваш проект двигался вперед с максимальной скоростью.

КОГДА НА ДОРОГЕ ДОЛЖНЫ ВАЛЯТЬСЯ ПОКРЫШКИ...

В нашей студии работает замечательный левел-дизайнер Хирокуза Яшура (Hirokazu

Yasuura), один из трех основных создателей игры Sonic The Hedgehog.

Приступая к работе, он сначала рисует мультфильм, который поясняет механику уровня, а потом с помощью воображения, листочка бумаги

и простого карандаша полностью создает карту уровня. Художники, отвечающие за проработку графических деталей, затем быстро делают модель уровня, которую Яшура вместе с другими сотрудниками тестирует и отдает на дальнейшую доработку.

На протяжении разработки всех игр серии Jak подобный подход к процессу создания уровней доказал свою состоятельность в вопросах надежности, качества и соблюдения сроков. Как бы то ни было, сразу же после завершения работы над картой мы тут же проверяли ее в игре, и к концу первого месяца разработки у нас уже был готов первый работающий уровень.

К тому времени уже был составлен предварительный график художественных работ и работ по созданию уровней. Так как размеры уровней весьма сильно варьировались, мы не следовали графику буквально. Наши художники весьма опытные и сами отвечают за своевременную сдачу работ. Мы крайне редко отказываемся от реализации каких-либо идей из-за нехватки времени. Такое может случиться лишь в исключительных случаях. Весь процесс создания графики у нас осуществляют очень квалифицированные и хорошо организованные сотрудники, которые высоко держат планку качества.

В полной мере разработка развернулась в ноябре. К зиме у нас уже были готовы три из семи арен и много различных уровней-трасс. Мы постоянно продолжали работать над художественными деталями. Это продолжалось до середины июля, когда была выпущена альфа-версия. Потом наше внимание было приковано к процессу подкачки уровней. Мы хотели, чтобы они загружались до того, как игроку предстояло их проходить. В результате (как это и было во всех играх серии Jak) нам пришлось изрядно поломать голову над тем, как подгружать каждый уровень в отдельности. Но в конце концов мы достигли того, что наши уровни включали в себя большое количество графики. Кроме того, некоторые из них были гибридными – состояли из различных локаций. Найденная нами схема загрузки работала практически без задержек.

ГЛАВНОЕ – ХОРОШЕНЬКО ПОДГОТОВИТЬСЯ!

Благодаря интенсивной работе программистов, художников и музыкантов разработка пошла очень быстро. Начав с базовых элементов игрового движка, мы довольно скоро добрались и до геймплея. Геймдизайнеры в Naughty Dog очень часто выступают в роли продюсеров, выполняя все присущие им обязанности. Нам нравится подобный подход, поскольку он позволяет команде быть в отличной форме и сконцентрироваться на создании игры, а не тратить время на бюрократические разборки или сваливать ответственность друг на друга.

К тому же мы стараемся как можно раньше формулировать задачи, которые необходимо решать (создание меню для интерфейса и прочее). Например, как только сделан меш уровня, все уже готово работать над реализацией графики и далее. Это позволяет нам достичь весьма высокой эффективности. Мы постоянно тестируем игру и стараемся поддерживать стабильную и своевременную обратную связь между художниками и программистами, так как в их случае коррективы в проект должны вноситься как можно скорее. Благодаря этому мы не расслабляемся и быстро достигаем нужного качества.

Если что-то идет не так, как запланировано, один из дизайнеров приходит на помощь другому члену команды, и вместе они ищут выход из сложившейся ситуации. Это – версия экстремального программирования от Naughty Dog. GOAL (LISP-подобный язык, разработанный нами, с помощью которого создавались игры серии Jak) вкупе с нашими собственными игровыми редакторами предоставляют нам неограниченный простор для экспериментов. С их помощью мы смогли легко и быстро достичь оптимальной игровой механики.

Со второго месяца разработки мы задействовали несколько исключительно опытных штатных тестеров. Это позволило получить множество откликов по поводу геймплея с самого начала игры и сделало процесс поиска багов наиболее эффективным. Мы постоянно работали над спецэффектами и звуком – так, чтобы в игре было полно деталей, блеска, звуковых эффектов, чтобы на протяжении всего действия лилась музыка. Мы так организовали хранение наших

аудиоданных, что в любой момент, если бы возникла такая необходимость, смогли полностью поменять звуковое сопровождение.

В конце концов, чем раньше у нас появилась бы возможность делать какие-либо выводы о нашей игре, тем быстрее мы могли бы двигаться к финишу.

Так как мы собрали основной игровой материал к весне-лету 2005 года, то у нас была возможность постоянно вносить в игру коррективы. Но, невзирая на то, что нами постоянно вносились значительные изменения в игровую механику, этот процесс всегда контролировался. Изменения вносились постепенно и небольшими порциями – только после одобрения членов команды и положительного отзыва тестеров. Мы неуклонно двигались к цели, избегая серьезных неприятностей, которые могли возникнуть тогда, когда в одном месте производилось слишком много неконтролируемых изменений.

Время от времени одна проблема заставляла нас снижать темпы – это совершенно необъяснимое увеличение размеров уже созданных уровней. Такое происходит весьма часто, когда вносятся множество мелких изменений в игру. Поскольку мы не использовали систему контроля версий, то нам было довольно трудно искать подобные ошибки.

Иногда мы делали паузы, чтобы напомнить самим себе, что же мы в конце концов хотим сделать и, конечно, для того, чтобы убедиться – все идет так, как нужно. Планировалось, что наша игра будет «взаимодействовать» с игрой *Daxter* PSP от компании *Ready at Dawn* (в каждом проекте была реализована возможность обмена данными с помощью USB-кабеля). Подобную «кросс-разработку» мы начали заранее, чтобы предотвратить возможные неожиданные сложности.

На протяжении всей разработки мы тщательно отслеживали все запланированные бонусы и тайники, так что нам было очень легко добавлять их. Также с помощью специальных инструментов мы осуществили локализацию игры непосредственно в ходе разработки. Кроме того, нам удалось на одном диске разместить версии на всех языках. К середине марта у нас было уже достаточно материала, чтобы сделать специальный пресс-релиз *Jak X: Combat Racing*. Несмотря на то, что содержание игры в дальнейшем еще улучшалось благодаря нашему особому подходу к разработке, удалось сделать демо-версию, которая произвела довольно сильное впечатление.

ПОСЛЕДНИЙ КРУГ!

Заранее, еще до выпуска в июле альфа-версии, мы расширили наш собственный отдел тестирования до 12 человек. Все они обладали знаниями, позволившими объяснить им, как компилировать игру и ежедневно записывать диски. Тем самым мы весьма облегчили жизнь программистам, которые чуть ли не каждую ночь занимались подобными вещами.

Благодаря тому, что мы сделали игру весьма гибкой, нам удалось показать ее лучшие стороны. До выхода бета-версии 11 ав-

густа мы организовали около пятнадцати презентаций. Мы могли позволить себе к концу проекта выкинуть пару уровней, если расписание поджимало. Это – величайшая роскошь для тех, кто работает в жанре экшн, где постоянная зависимость сотрудников студии друг от друга служит в подобных ситуациях причиной сильной головной боли.

К всеобщему восторгу *Jak X* работала весьма гладко, однако одна неприятность все же всплыла. Дело в том, что размер игры оказался намного больше, чем планировалось. Из-за этого возникли сложности при распространении тестовых материалов. Оглядываясь назад, скажу, что нам следовало бы делать более подробный план и более скрупулезно ему следовать, чтобы не возникло сложностей в конце проекта и можно было каждый уровень довести до подобающей степени совершенства.

С апреля 2005 года мы начали активное тестирование. Оно проходило в Фостере, в офисе компании *Sony Computer Entertainment of America*. Полученные данные мы записали на карты памяти *PlayStation 2*. Эти карты впоследствии были использованы для того, чтобы определить, насколько успешно игроки преодолевают трудности, и, таким образом, для настройки сложности игры.

Сетевой старт бета-версии в июне был чрезвычайно успешным. На PlayStation.com находилась доска объявлений, которая обеспечивала нам бесценную обратную связь с игроками. Если мы желали выяснить, что в игре интересно, а что нет, то всегда могли положиться на нашу команду тестеров.

Активное тестирование продолжалось вплоть до последней недели перед выходом бета-версии. У нас появилось много друзей в сети, которые могли поделиться свежими идеями.

И, наконец, тесное сотрудничество между нашей командой, нашими замечательными продюсерами из *Sony* и нашим отделением контроля качества стало определяющим фактором того, что проект без особых препятствий достиг заключительных стадий разработки. Все художники и программисты работали не покладая рук, чтобы отшлифовать содержание и выявить последние баги. Качественная организация, непринужденная и легко доступная коммуникация, чрезвычайная сплоченность в коллективе позволили нам без особых проблем преодолеть этап контроля качества и завершить проект точно к сроку – к сентябрю.

ОГЛЯНИСЬ!

Мы надеемся, что этот обзор разработки *Jak X: Combat Racing* будет вам полезен, потому что вы как бы переживете то, с чем нам пришлось столкнуться самим. Мы верим, что эта статья поможет делать игры быстро и эффективно. То, что мы делаем в нашей студии, мы делаем лишь потому, что любим игры и любим тех, кто в них играет. Мы чертовски счастливы, что у нас такая замечательная команда! Это именно она привела *Jak X* к успеху – спасибо, ребята! Мы с нетерпением ждем 2006 года, который принесет нам новые игровые консоли и еще много чего хорошего.



Хирокуза Яшура делает эскиз уровня на бумаге, которая затем передается художникам, создающим графические детали





АНДРЕЙ АКСЕНОВ

>> ДОЗА ПИКСЕЛЕЙ

ШЕЙДИНГ
В «МАГИИ КРОВИ»

Андрей Аксенов — технический директор компании «SkyFallen Entertainment». В ее составе он работал над созданием ролевой игры «Магия Крови». Любые вопросы и пожелания адресуйте ему на shodan@shodan.ru

ШЕЙДИНГ «МАГИИ КРОВИ» ЗА ВРЕМЯ РАЗРАБОТКИ СТАЛ ДОВОЛЬНО ОБЫЧЕН — В ОТЛИЧИЕ ОТ 2002 ГОДА, КОГДА ПРОЕКТ ТОЛЬКО СТАРТОВАЛ. Сейчас уже все знают, что такое бамп, спекуляр, динамические тени и зачем нужно покупать видеокарту с поддержкой шейдеров. Поэтому речь пойдет не о том, как реализовать очередную версию бампа, а о том, как это сделать максимально эффективно и при минимальных ресурсах — то есть при использовании шейдеров версии 1.1.

Я надеюсь, что эта статья будет интересна читателям не только как исторический экскурс. Во-первых, пользователей с видеокартами, не поддерживающими шейдеры версии 2.0 или значительно замедляющимися при работе в этом режиме, все еще достаточно — по данным Valve Steam Survey за март 2006 года (Steam06), примерно 25%. Во-вторых, трюки, описанные в статье, могут быть применены и для шейдеров 2.0 и выше.

Итак, материал, в основном используемый в «Магии Крови», представляет собой комбинацию попиксельного освещения от одного направленного источника (солнца) и повертексного диффузного освещения от нескольких omni-источников¹. Для солнца используется модель освещения Блина:

$$\text{Pixel} = \text{Texture}_{\text{Diffuse}} \times (\text{Color}_{\text{Diffuse}} \times (N \cdot L) \times K_{\text{Shadow}} + \text{Color}_{\text{Ambient}}) + \text{Texture}_{\text{Specular}} \times \text{Color}_{\text{Specular}} \times (N \cdot H)^{\text{SpecularPower}} \times K_{\text{Shadow}}$$

(1)

Здесь вектор нормали N берется попиксельно из карты нормалей; K_{Shadow} — па-

раметр, определяемый для каждого пикселя по карте теней, который равен 0, если пиксель находится в тени, и 1 в противном случае; L — нормализованный вектор направления источника света; H — вектор-полусумма² (half-vector):

$$H = \text{normalize}(L + \text{normalize}(\text{Camera} - \text{Position}))$$

(2)

Для реализации такого материала при помощи шейдеров 1.1 теоретически необходимо накладывать:

- диффузную текстуру;
- карту нормалей (bump map);
- карту отраженного цвета (specular map);
- карту теней (shadow map);
- нормализующую кубическую карту (normalizer cube map) для вектора H .

Остановимся чуть подробнее на последней. Если вектор H не нормализовать в пиксельном шейдере, то появляются артефакты — отраженный свет выглядит не так, как должен. Несмотря на то, что в пиксельных шейдерах 1.1 нет ни инструкции pmg, ни инструкции rsq, нормализовать вектор все равно можно (при помощи приближенных вычислений для корня длины). К несчастью, в этом случае на одну только нормализацию вектора H уйдет непоправимо много инструкций. Поэтому приходится использовать нормализующую карту. Кажется, что за счет одного только количества текстур для шейдеров 1.1 мы уже вынуждены делать 2 прохода отрисовки. Однако при помощи ряда трюков все от-

лично укладывается в 1 проход. Посмотрим, как именно.

Во всех последующих примерах кода предполагается, что наше оборудование аппаратно поддерживает карты теней — то есть это какая-то из видеокарт NVidia, на-

чиная с GF3. Для видеокарт ATI потребуются поддержка пиксельных шейдеров версии 1.4, на которой наложение карты теней несложно реализовать вручную.

ТРЮК 1.
ЗАПЕКАНКА ИЗ ТЕКСТУР

Если использовать в карте отраженного цвета (specular map) одни только оттенки серого, то ее можно поместить в альфа-канал карты нормалей. Для того чтобы частично компенсировать потерю цветности, введем в модель освещения параметр ColorSpecular, на который будем умножать прочитанное в альфа-канале значение.

Поскольку обычно для наложения карты нормалей и карты отражений используется один и тот же набор текстурных координат, такое совмещение текстур не должно вызвать особых сложностей.

С помощью этого трюка мы сводим число одновременно накладываемых текстур до 4 и помещаемся в лимиты 1.1 по текстурам. Следующая проблема состоит в том, что мы можем использовать только 8 инструкций, а для реализации уравнения (1) «в лоб» нам потребуется около 10, даже если отвести на возведение в степень только 2 умножения.

¹ Omni-light — точечный источник света, одинаково излучающий свет во всех направлениях.

² Half-vector — вектор, используемый в модели освещения Блина.

ТРЮК 2. ДВА ПИКСЕЛЬНЫХ ШЕЙДЕРА В ОДНОМ ЗА ТУ ЖЕ ЦЕНУ

Пиксельные шейдеры 1.1 могут одновременно выполнять 2 инструкции, если одна из них записывает результат в компоненты RGB (для нее используется векторный конвейер), а другая скалярная и записывает результат в канал A (соответственно используется отдельный скалярный конвейер). При помощи такого спаривания мы можем увеличить число инструкций до 16.

Вычисление диффузного освещения в основном сводится к работе с 3-компонентными цветами и векторами – так что векторный конвейер будет занят именно этим. Однако ощутимая часть расчетов отраженной компоненты освещения может быть выполнена и на скалярном конвейере (Листинг 1).

Отметим, что расчет отраженного освещения в общем случае требует большего числа операций по сравнению с диффузным, поэтому запускать его следует как можно раньше.

ЛИСТИНГ 1

```
ps_1_1

// v0 = фоновый свет (ambient)
//   + собственная светимость (luminosity)
//   + повертексное освещение

// v1 = вектор L в тангенциальном пространстве
//      (tangent space)

#define SPEC c6 // c6 = отраженный (specular)
// цвет материала
#define DIFF c7 // c7 = диффузный (diffuse) цвет
// материала

tex t0 // карта теней
tex t1 // смешанная карта нормалей и отраженного
// цвета
tex t2 // нормализованный вектор-полусумма N
tex t3 // диффузная текстура

dp3_sat r1.rgb, t1_bx2, t2_bx2 // (1) (N.H)

dp3_sat r0.rgb, t1_bx2, v1_bx2 // (2) (N.L)
+mul_sat r0.a, r1.b, r1.b // (2) (N.H)^2
```

```
mul r0.rgb, r0, t0 // (3)
(N.L)*shadow

+mul r0.a, r0.a, t0.b // (3)
((N.H)^2)*shadow

mad_sat r0.rgb, r0, DIFF, v0 // (4)
(N.L)*shadow*diffcol+ambi

+mul_sat r0.a, r0.a, r0.a // (4)
((N.H)^4)*shadow

mul_sat r0.rgb, r0, t3 // (5)
diffmap*diffighting

+mul_sat r0.a, r0.a, t1.a // (5)
((N.H)^4)*shadow*specmap

mad_sat r0.rgb, r0.a, SPEC, r0 // (6) result
+mov r0.a, t3.a // (6) diffuse
map alpha
```

Этот шейдер использует только 6 слотов инструкций, так что у нас остается в запасе 2 слота для дополнительных расчетов. В частности, ниже будет показано, как вместо фиксированной степени 4 использовать для расчета отраженного света произвольно взятую степень

ТРЮК 3. НОВЫЙ ШАМПУНЬ ДЕЛАЕТ ВАШИ ТЕНИ ОБЪЕМ- НЕЕ НА 37%!

Шейдер из листинга 1 дает довольно неинтересную картинку в затененных местах. Причина в том, что единственной ненулевой составляющей цвета по формуле (1) оказывается модулированная фоновым цветом диффузная текстура, из-за чего тени получаются плоскими (не учитывается карта нормалей).

Чтобы избавиться от этой проблемы, введем коэффициент прозрачности теней и добавим его в формулу (1):

$$Pixel = Texture_{Diffuse} \times (Color_{Diffuse} \times (N \cdot L) \times DiffK_{Shadow} + Color_{Ambient}) + Texture_{Specular} \times Color_{Specular} \times (N \cdot H)^{SpecularPower} \times SpecK_{Shadow} \quad (3)$$

где

$$\begin{aligned} DiffK_{Shadow} &= K_{Shadow} \times (1 - DiffTransp) + DiffTransp, \\ SpecK_{Shadow} &= K_{Shadow} \times (1 - SpecTransp) + SpecTransp, \\ DiffK_{Shadow}, SpecK_{Shadow} &\in [0,1]. \end{aligned} \quad (4)$$

Коэффициенты DiffKShadow и SpecKShadow настраиваются художниками для каждого материала. Они отвечают за то, какая доля диффузной и отраженной составляющей освещения будет добавлена к фоновому освещению на затененных участках. Это дает художникам возможность убирать «плоскую» картинку в тени. Сам код шейдера изменяется лишь незначительно.

ЛИСТИНГ 2

```
#define SPECK c4 // c4.rgb=1-DiffTransp,
c4.a=DiffTransp
#define DIFFK c5 // c5.b=1-SpecTransp,
c5.a=SpecTransp

dp3_sat r1.rgb, t1_bx2, t2_bx2 // (1)
(N.H)

dp3_sat r0.rgb, t1_bx2, v1_bx2 // (2)
(N.L)
+mul_sat r0.a, r1.b, r1.b // (2)
(N.H)^2

mad r1.rgb, t0, DIFFK, DIFFK.a // (3) diff-
shadow
+mad r1.a, t0.b, SPECK.b, SPECK.a // (3)
specshadow

mul r0, r0, r1 // (4)
(N.L)*diffshadow // (4)
((N.H)^2)*specshadow

// слоты 5-7 аналогичны слотам 4-6 в листинге 1
```

Позэкспериментировав с этим шейдером, мы выяснили, что напрямую можно контролировать только SpecKShadow – потому что изменение картинки за счет DiffKShadow не очень незаметно. Поэтому

ДОЗА ПИКСЕЛЕЙ

в итоге мы остановились на том, что художникам доступен только один параметр SpecKShadow, значения которого лежат в интервале [0.5, 1], а DiffKShadow вычисляются следующим образом:

$$\text{DiffK}_{\text{Shadow}} = 2 \times \text{SpecK}_{\text{Shadow}} - 1, \quad (5)$$

причем наличие модификатора _bx2 делает это вычисление бесплатным.

ТРЮК 4. ДА БУДЕТ (ЕЩЕ) СВЕТ!

Мы получили пиксельный шейдер, который за один проход осуществляет поликсельное диффузное и отраженное освещение с учетом теней. Но есть еще и вершинный шейдер, который можно использовать для расчета какого-то более сложного вертексного освещения.

Добавим в вертексный шейдер расчет диффузного освещения за счет настолько большой кучи omni-источников, насколько получится (реализацию spotlight источников, которые отличаются одним дополнительным коэффициентом затухания в зависимости от угла между точкой и направлением источника, оставим как упражнение для читателей). Формула для вычисления освещения от одного источника имеет следующий вид:

$$\begin{aligned} \text{Lighting} &= \text{OmniColor} \times \text{Diffuse} \times \text{Attenuation}, \\ \text{Diffuse} &= \text{Clamp}(N \cdot \text{normalize}(\text{OmniPos} - \text{Pos}), 0.1), \\ \text{Attenuation} &= \text{Clamp}\left(1 - \frac{(\text{OmniPos} - \text{Pos})^2}{\text{OmniRange}^2}, 0.1\right) \end{aligned} \quad (6)$$

Вычисления «в лоб» можно уместить где-то в 10-13 инструкций вертексного шейдера. Однако если сгруппировать источники света по 4, то всю обработку этой группы можно осуществить всего за 25 инструкций.

ЛИСТИНГ 4

```
// OMNIRANGES = ( 1.0f / (omni[i].range^2) )
add r3, -v0, OMNIPOS[0] // L0 = center0-point
```

```
add r4, -v0, OMNIPOS[1] // L1 = center1-point
add r5, -v0, OMNIPOS[2] // L2 = center2-point
add r6, -v0, OMNIPOS[3] // L3 = center3-point
dp3 r7.x, r3, r3 // dist0^2
dp3 r7.y, r4, r4 // dist1^2
dp3 r7.z, r5, r5 // dist2^2
dp3 r7.w, r6, r6 // dist3^2
rsq r8.x, r7.x // 1/dist0
rsq r8.y, r7.y // 1/dist1
rsq r8.z, r7.z // 1/dist2
rsq r8.w, r7.w // 1/dist3
mul r7, r7, -OMNIRANGES // -
dist_i^2/range_i^2
max r7, r7, -ONE // clamp(-
dist_i^2/range_i^2,-1.0)
dp3 r2.x, v1, r3 // N dot L0
dp3 r2.y, v1, r4 // N dot L1
dp3 r2.z, v1, r5 // N dot L2
dp3 r2.w, v1, r6 // N dot L3
mul r8, r8, r2 // N dot normalize(L_i)
max r8, r8, ZERO // clamp N dot L
mad r8, r8, r7, r8 // clamp(N dot L_i)*attn_i
mul r9, r8.x, OMNICOLOR[0] // r9 = omni0.dif-
fuse
mad r9, r8.y, OMNICOLOR[1], r9 // r9 += omni1.dif-
fuse
mad r9, r8.z, OMNICOLOR[2], r9 // r9 += omni2.dif-
fuse
mad r9, r8.w, OMNICOLOR[3], r9 // r9 += omni3.dif-
fuse
```

Ключевая идея состоит в том, чтобы векторизовать операции над отдельными скалярными величинами – примером может служить почти что каждая инструкция mul, mad или max в приведенном выше коде. Отметим, в частности, что для лучшей векторизации в константу OMNIRANGES сгруппированы величины, обратные квадрату радиуса источника.

Далее в вершинном шейдере, фрагмент которого приведен, освещение от всех групп omni-источников складывается вместе, складывается с фоновым и собственной светимостью материала и помещается в регистр oD0. Пиксельный шейдер не нуждается в модификации.

Несмотря на необходимость рассчиты-

вать в вертексном шейдере скелетную анимацию для самой вершины и базиса тангенциального пространства, туман, текстурную анимацию и так далее, с применением приведенного фрагмента кода ограничения на размер шейдера 1.1 в 128 инструкций вполне хватает для того, чтобы одновременно обчислить все это и еще 12 omni-источников в придачу.

ТРЮК 5. ВСПОМИНАЕМ ПРО ОДИН БЛЕСТЯЩИЙ ТРЮК

Этот трюк не нов и придуман не мною, так что, честно говоря, эту часть статьи можно назвать напоминанием.

В нашем пиксельном шейдере используется фиксированная степень отражения (степень, в которую возводится скалярное произведение N dot N), равная 4, реализованная при помощи двух инструкций – последовательных умножений числа на само себя. Однако благодаря блестящему исследованию Philippe Beaudoin разлива 2002 года (Beaudoin02) мы можем в то же самое количество инструкций реализовать приближенный расчет других степеней.

Основная идея состоит в том, чтобы заменить честное возведение в степень аппроксимацией такой функцией, которую легко реализовать с помощью инструкций пиксельного шейдера. Один из хороших вариантов вида аппроксимации, который мы будем использовать, таков:

$$x^n \approx 16 \times (Ax + B)^2, \quad (7)$$

$$A, B \in [0.1]$$

где коэффициенты A и B вычисляются заранее специальной программой с циничной целью уменьшения погрешности. При использовании функции данного (квадратичного) вида максимальная абсолютная погрешность лежит в пределах 0.04-0.06; если необходима более высокая точность, можно использовать полиномы более высоких степеней. За подробной информацией, другими видами аппроксимирующих функций и исходным кодом см. (Beaudoin02).

После применения этого трюка наш многорадальный пиксельный шейдер принимает следующий окончательный вид.

ЛИСТИНГ 5

```
#define POW c3 // c3.b=B, c3.a=A, for m=2. cm.
[Beaudoin02]

dp3_sat r1.rgb, t1_bx2, t2_bx2 // (1) (N.H)

dp3_sat r0.rgb, t1_bx2, v1_bx2 // (2) (N.L)
+mad_x4_sat r0.a, r1.b, POW.a, POW.b // (2)
(N.H)*A+B

mul_x4_sat r1.rgb, r0.a, r0.a // (3) (N.H)^n
+mad r1.a, t0.b, SPECK.b, SPECK.a // (3) spec-
shadow

mul_sat r0.rgb, r0, r1_bx2.a // (4) (N.L)*diffshadow
+mul_sat r0.a, r1.b, r1.a // (4) ((N.H)^n)*specshadow

mad_sat r0.rgb, r0, DIFF, v0 // (5) (N.L)*shad-
ow*diffcol+ambi
+mul_sat r0.a, r0.a, t1.a // (5) ((N.H)^n)*shad-
ow*спекmap

mul_sat r0.rgb, r0, t3 // (6) diffmap*difflighting

mad_sat r0.rgb, r0.a, SPEC, r0 // (7) result
+mov r0.a, t3.a // (7) diffuse map alpha
```

Единственное, что еще осталось ска-
зать, — константы A и B на самом деле
можно даже не рассчитывать специально
обученной программой, а давать на конт-

роль художникам — эти параметры ведут
себя достаточно интуитивно понятно для
того, чтобы можно с ними было работать
напрямую.

ЧТО БЫЛО ДАЛЬШЕ?

Продемонстрированные трюки вовсе не
ограничены ни шейдерами версии 1.1, ни
аппаратной поддержкой карт теней.

Довольно просто портировать приведен-
ный пиксельный шейдер на версию 1.4 —
при отрисовке в карту теней упаковывать
информацию о глубине в ARGB8 цвет и
распаковывать его при наложении карты.
Используя две дополнительные текстуры и
дополнительные инструкции, несложно
расширить шейдер, например, наложени-
ем кубической карты окружения с учетом
специальной маски или карты нормалей —
при этом все еще в один проход.

Описанные трюки также можно приме-
нить и на DX9-оборудовании. Это очевидно
для вертексных шейдеров, но на самом
деле справедливо и для пиксельных — из-
вестно, например, что чип R300 внутрен-
ним образом поддерживает независимые
векторный и скалярный конвейеры и для
пиксельных шейдеров 2.0, то есть проде-
монстрированное в статье спаривание
инструкций может быть использовано для
оптимизации и в версии 2.0.

На самом деле у нас все еще осталась
пара свободных инструкций даже в шейде-
ре версии 1.1, так что есть место для улуч-
шений. Возьмем, например, карту цвета
отражений, которую мы перевели в оттен-

ки серого и упаковали в альфа-канал кар-
ты нормалей. Можно модулировать ее не
просто цветом, а диффузной текстурой.
Также можно передавать в шейдер два
цвета, а значение текстуры использовать
для интерполяции между ними. Или упоко-
вать еще один набор чисел в альфа-канал
диффузной текстуры, если он не задей-
ствован, и использовать его для восста-
новления еще большего количества дета-
лей карты цвета отражений.

И в конце концов... всегда остается ва-
риант с двумя проходами.

ОТХОДНАЯ

В статье рассмотрена реализация попик-
сельного освещения с тенями от одного
направленного источника и повертексного
диффузного освещения от 4 до 12 omni-
источников — все это за 1 проход на шей-
дерах версии 1.1. Описанная реализация
успешно применяется в игре «Магия Кро-
ви», а также в ряде других проектов (вклю-
чая «Не время для драконов» и «Санитары
подземелий»), использующих ее движок.

Описанные в статье методы в основном
были разработаны автором в 2003 году во
время работы над проектом «Магия Кро-
ви» в компании Skyfallen Entertainment. Не-
большие дополнительные оптимизации
шейдеров регулярно производились с
2003 по 2005 годы включительно.

Автор хотел бы особо поблагодарить
Бориса Баткина за его бесценные советы
по программированию графики в целом и
оптимизации шейдеров в частности.

ССЫЛКИ

[Steam06] Valve's Steam Survey, <http://www.steampowered.com/status/survey.html>

[Beaudoin02] Beaudoin P., Guardado J., «A Non-Integer Power Function on the Pixel Shader»,
http://www.gamasutra.com/features/20020801/beaudoin_01.htm



РУСЛАН АБДИКЕЕВ

>> ВНУТРЕННИЙ ПРОДУКТ

A ship in harbor is safe -- but that is not what ships are for.

John A. Shedd

IT HAS BEGUN...

ОБЫЧНО ВСЕ НАЧИНАЕТСЯ УТРОМ, С НЕВИННОГО ГЛЮКА, С МИМОХОДОМ БРОШЕННОЙ ФРАЗЫ ТЕСТИРОВЩИКА:

- Кстати, мне понравился дым от объекта, когда у него броня на исходе. Классная идея.

- Ммм, а что с дымом?

- Ну, раньше кровавый дым был, а теперь – наоборот: как здоровья стало мало – дым начинает ярко светиться, будто перед взрывом.

Выражение лица Юры (начальника QA) начинает тянуть баллов на пять по шкале Рихтера (глаза прищурены, в бровях – удивление, руки теребят «Список изменений»):

- И давно у нас такое нововведение в дизайне?

Тестировщик, конечно же, не понимает, с чего бы это так все напряглось.

А у меня в голове – только одно: лишь бы это было изменение в дизайне, лишь бы я о нем просто забыл, лишь бы...

Ведь все прощу. Даже не покалечу. К тому же идея действительно может быть хороша.

- Не знаю, Юра, сейчас подниму change history.

Но противное ощущение падения где-то в низу живота остается.

Потому как изменений в дизайне точно не было, а до code complete – меньше недели.

И предательское «на всякий случай» вырывается само собой:

- Знаешь, Юра, на всякий случай, – глянь release билд на предмет цветов...

Через час я уже точно знаю, что никаких изменений никто не делал.

Ни в коде particle systems, ни в эффектах, ни в настройках, ни в гейм-дизайне.

Это место вообще уже давно никто не трогал.

Ну, и начало конца – звонок из QA:

- Проверили release билд – дым моргает всеми цветами радуги.

Собственно, это только часть проблемы.

В баг трекере – пять новых showstoppers. Появились «неубиваемые» объекты.

Появились «невыполнимые» миссии. Полный, так сказать, mission failure.

В реальной жизни все именно так и происходит.

В данном конкретном случае – хоть понятно, куда breakpoint ставить, – и то хорошо.

Как показало расследование – цвет дыма оказался 0xDDDDDDDD, ну а дальнейший ход событий вы сможете восстановить сами.

ДАЕШЬ ПРОФИ!

Все мы любим работать в команде профессионалов, с людьми, у которых за плечами серьезный опыт и несколько выпущенных игр. С ними не особенно нужны формальные coding style guide и code review: как правило, хорошие программисты в команде и так непрерывно общаются, читают одни и те же книжки и просматривают sweng и rsdn. Здесь уж скорее нужны wiki со списком интересных цитат, ссылок, фрагментов кода. Ну а ошибки – они чаще на уровне взаимопонимания, чем в технической реализации.

Но по мере роста коллектива вам обязательно дадут в подчинение и обучение молодых коллег, не имеющих практически

никакого опыта разработки приложений.

Стандарт кодирования (coding style guide) – суровая необходимость, если мы работаем с теми, кто только начинает свою профессиональную карьеру. Стандарт кодирования не заставит людей программировать лучше. Единственный способ обучения идиоматике языка и его опасным местам – смотреть много чужого кода, читать книжки, слушать чужие советы и много программировать. Другими словами – реальный боевой опыт в реальном проекте и ваш личный пример.

Стандарт кодирования – это очень простое средство, позволяющее упростить совместную работу над исходниками (там, где это необходимо) и создать безопасную «песочницу» (safe sandbox).

В рамках этой «песочницы» программист просто не сможет допустить особенно опасных ошибок.

Если двинуться чуть дальше – «песочница» определяет не только безопасность, но и специализацию программиста. В играх (из-за обилия data-driven решений) «песочница» может быстро перестать кодом вообще, и превратиться в один из инструментов разработки контента.

Совместно со стандартом кодирования должна быть принята практика вычитки кода (code review).

Один из самых тяжелых вопросов по code review – когда именно его делать и как часто.

Кто-то это делает при каждом check in, кто-то – аккумулирует набор и делает вычитку раз в неделю.

Важно, чтобы ведущим программистом были четко определены момент, способ и участники code review.

Подавляющее большинство «тупых» (но от этого – не менее болезненных) ошибок происходит из-за элементарной невнимательности или забывчивости.

Стандарт кодирования тренирует привычку, а code review – позволяет отловить явные проблемы.

Ну и завершающий элемент формирования базовой культуры программирования – практика использования компилятора и отладчика (включая внештатные средства типа bounds checker).

Следует хорошо себе представлять, в чем ваши инструменты могут и не могут вам помочь.

Сюда же относятся рекомендации (если они для вас применимы вообще и не окажутся overkill):

- используйте минимум два компилятора (для кого-то – это требование) и минимум два STL,
- запускайте игру под минимум двумя сильно отличающимися locale,
- запускайте как debug-, так и release-билд и т.п.

Следование этим простым рекомендациям не только позволит исключить ряд типичных проблем (скажем, указатели вместо итераторов (STL), запятые в числах вместо точек (locale), невалифицированные указатели на члены (компилятор), не по делу переносимый код и т.п.), но и быстро повысит кругозор ваших подопечных.

В случае C++ (как очень опасного языка), жить без стандарта кодирования и code review фактически нельзя.

И первое, что должен описывать стандарт – это не «где ставить скобки», а как избежать классических проблем C++: неопределенного поведения, проблем с памятью, с ресурсами и с производительностью.

Причем – в очень тупых и простых терминах: «не пиши так, пиши вот так», с объяснением и примерами.

Много копий сломано по поводу того,

насколько проверяемыми и формализуемыми должны правила в стандарте кодирования. Довольно распространенное мнение – все правила в стандарте кодирования должны допускать их автоматическую проверку. Я не согласен.

Мое личное мнение: место полностью формализуемых правил – в полностью автоматических инструментах и документации к ним, а не в стандарте кодирования. Языки или инструменты, которые жестко фиксируют синтаксис, исключают священные войны «пробел после скобки» и «табы против пробелов» наиболее радикальным способом из всех возможных – некорректный код просто не компилируется и отвергается. На мой взгляд – это не всегда приятное, но всегда достаточно сильное средство.

Смысл же стандарта кодирования, – в описании best practices, которые редко когда можно формализовать.

Очень важно соблюсти разумный баланс между чрезмерно жестким стандартом кодирования и чрезмерно мягким. Любой стандарт должен содержать четко сформу-

лированную мысль об отпущении грехов при явном его нарушении. Но каждое нарушение стандарта кодирования должно быть четко отражено в комментариях к коду, и, при постоянном повторении ситуации, должно попасть в стандарт кодирования как принятая практика при специфических исключительных обстоятельствах.

Очевидно, что стандарт кодирования – это всегда живой документ, развивающийся вместе с программистами, для которых он предназначен. Если коллеги неоднократно совершили одну и ту же ошибку и есть риск ее повторного возникновения, – имеет смысл занести ее в стандарт кодирования.

Главное в стандарте кодирования – не чтобы он был, а чтобы его читали.

Не ударяйтесь в сферическо-вакуумный стандартный C++ (в котором std::string::swap из слов «спаниель»+«форт» после обмена делает «торф»+«апельсин»¹). Соотносите практичность, конкретные реализации и возможности вашей «песочницы».

Для кого-то совершенно неважна коррект-

НЕИНИЦИАЛИЗИРОВАННЫЕ ДАННЫЕ CODING GUIDELINES:

- » При компиляции должен стоять максимальный уровень предупреждений (MSVC: /W4).
- » При компиляции debug-билда должны быть включены runtime проверки (MSVC: /RTC1 /RTCс).
- » Любая переменная в функции должна иметь явный инициализатор и определяться в месте использования.
- » Где возможно, члены класса должны инициализироваться в списке инициализаторов, а не в теле конструктора.
- » В высокоуровневом классе не должно быть деструктора, операторов присваивания и конструкторов копирования.
- » В конструкторе высокоуровневого класса должны перечисляться только члены, которые явно зависят от параметров конструктора или от специфики класса. Все остальные члены должны иметь корректные конструкторы по умолчанию.
- » В местах, где вы очень боитесь того, что вам передадут неопределенные значения – проверяйте их функциями core::check_if_pointer_is_readable<T>() и core::check_if_value_is_initialized<T>().
- » При генерации бинарных файлов убедитесь, что в них не видно зон, забитых магическими заполнителями.
- » Любой нестатический POD-член класса должен быть проинициализирован во всех конструкторах. Альтернативно, следует использовать const T для POD-членов. Альтернативно, следует использовать core::explicitly_initialized<T> вместо T для POD-членов. В крайнем случае, следует использовать boost::value_initialized<T> вместо T для POD-членов.
- » Переменные не должны «повторно использоваться» для разных целей – заводите разные переменные.
- » Использование C-style массивов запрещено, вместо них следует использовать std::vector или boost::array.
- » Любые API, которые возвращают значения через ссылки, должны быть исключены (переделаны) или обернуты в функции, возвращающие одно композитное значение (структуру, boost::tuple и т.п.).

¹ std::string::swap не гарантирует, что порядок символов в строках после обмена останется прежним (21.3.5.8, DR 535).

ВНУТРЕННИЙ ПРОДУКТ

³ Aliasing (в данном контексте) – возможность получить доступ к одному и тому же объекту, используя разные имена (через указатели и ссылки). Может привести к невозможности полноценного анализа и выполнения ряда оптимизаций. В C99 допустимо использовать явный хинт (restrict), обещающий компилятору, что других псевдонимов у объекта нет.

ность работы со случайными числами. Для кого-то – напротив, это вопрос вполне реальных денег. Реальный вопрос – реальная надежность, и кто-то будет читать стандарт², изучать исходники std::random_shuffle, читать «Improvements to TR1 RNG» и предлагать песочницы, в которых люди не будут сами масштабировать случайные числа.

Есть, впрочем, общая для любой компании часть, с которой, прежде всего, сталкиваются новички.

чивости, не предотвращенной компилятором.

Получить неинициализированную нестатическую переменную в функции на современных компиляторах довольно тяжело. Например, в Visual C++ на помощь сразу приходят:

- предупреждения C4700 (уровень 1) и C4701 (уровень 4),
- флаг /RTCu, который выдаст в runtime ошибку, если используется неинициализированный скаляр,
- магическая константа 0xCCCCCCCC

Из-за aliasing Visual C++ не предупредит вас о неинициализированной переменной.

Обязательно попробуйте этот пример кода – с aliasing и без – на ваших любимых компиляторах.

Это позволит вам узнать, от чего вас не защитят, и какую диагностику в compile- и run-time ожидать.

Вообще, для различного рода магических констант MSVC, RTL, bounds checker и т.п. и системных вызовов типа Win32

МАГИЧЕСКИЕ ЗАПОЛНИТЕЛИ MICROSOFT VISUAL C++

Заполнитель	char, Win1251	int	float	Что означает
0xCCCCCCCC	-52 'M'	-858993460	-1.0737418e+008	Неинициализированные данные на стеке
0xCDCDCDCD	-51 'H'	-842150451	-4.3160208e+008	Неинициализированные данные в heap
0xDDDDDDDD	-35 'Э'	-572662307	-1.9983972e+018	Данные в уничтоженном блоке heap
0xFDFDFDFD	-3 'э'	-33686019	-4.2201683e+037	Защитная область с обеих сторон блока heap

⁴ Полное определение POD (plain old data) типов можно найти в стандарте (9/4, 8.5.1/1, 3.9/10). Грубо говоря – это все скалярные типы (в том числе – перечисления и указатели) и рекурсивно построенные от них массивы и структуры, не содержащие ссылок, не-public членов и объявленных пользователем конструкторов, деструкторов, операторов присваивания, базовых классов и виртуальных функций. Если бы не константные члены и новинки C99 – сюда попадала все, что есть в C.

Давайте на секунду представим себе, что мы живем в идеальном мире, где много-много памяти и очень-очень быстрые процессоры. В мире, где мы никогда не вызываем delete/delete[]/free() и вообще никогда не освобождаем ресурсы. Где логика работы игры не зависит от того, жив объект или его уже удалили.

Какая беда будет подстерегать нас в этом удивительном C++ сразу же у входа в Эдем?

Неинициализированные данные опасны непредсказуемостью поведения программы и наведенными ошибками. В этом смысле проблемы очевидны всем. Менее очевидно, что можно сделать, чтобы их предотвратить. Мне приходилось неоднократно встречать в стандартах кодирования коронную фразу: «Запрещено использование неинициализированных данных». Довольно смешное требование, проще уж тогда сразу написать «Запрещено допускать ошибки». Ведь не из злого же умысла программист это делает.

Откуда же берутся в C++ неинициализированные данные?

Чаще всего – из-за элементарной забыв-

(для автоматических переменных) в отладочной версии.

Максимальный уровень предупреждений при компиляции должен стоять всегда: легче отключить назойливое предупреждение, чем пропустить что-то potentially важное.

То же самое касается в отладочных билдах флага /RTCu (точнее, всего набора /RTCc и /RTC1): вреда немного, а если сработают – вы сэкономите себе время и нервы.

Следует помнить, что эти меры – не панацея, а просто помощь.

Например, aliasing³ объекта через указатели или ссылки часто блокирует анализ компилятора:

```
int main()
{
    int a;
    int *b=&a; // если убрать эту
    строку, компилятор возмутится, а /RTCu –
    уронит.
    return a;
}
```

IsBadReadPtr(), имеет смысл завести вспомогательные отладочные функции, которые могут проверить, что указатели – валидны, а данные не являются магической константой-заполнителем.

(К вопросу об использовании CRT debug heap и bounds checker мы вернемся ниже.)

В особенных случаях (когда проблема появлялась не раз и не два) следует явно описать эти функции в стандарте кодирования, чтобы люди имели возможность по меньшей мере assert ставить.

Также следует порекомендовать людям проверять (хотя бы визуально) сгенерированные программой бинарные файлы. Очень часто в них можно встретить целые россыпи 'HHHH', 'MMMM' и 'EEEE', и эти «мычание» и неуверенность в высказываниях означают, что записали мусор, в том числе – неинициализированные данные и padding.

Возможно, это как раз то, что вы хотели – но стоит перепроверить.

В конце концов, если ваши коллеги уви-

² Стандарт разрешает std::random_shuffle использовать генератор случайных чисел, но не обязывает это делать (25.2.11, DR 552).

дят магическую константу вместо значения – они хоть будут знать, в каком направлении думать.

Итак, в свете улучшения качества компиляторов, существуют следующие причины получить неинициализированные данные:

- » иллюзия, что объект проинициализирован,
- » отделение объявления от инициализации и использования при объявлении класса,
- » использование C-style массивов,
- » отделение объявления от инициализации и использования из-за плохого дизайна API,
- » обращение к подобъектам во время конструирования полного объекта.

Если объявление переменной находится в месте ее использования в функции (чего требует любой стандарт кодирования), то вменяемый программист просто не сможет забыть ее проинициализировать, а компилятор ему поможет.

РАССМОТРИМ ПРОБЛЕМНЫЕ ЗОНЫ ПОДРОБНЕЕ.

ИЛЛЮЗИЯ, ЧТО ПЕРЕМЕННАЯ ПРОИНИЦИАЛИЗИРОВАНА

Иллюзия чаще всего бывает вызвана использованием так называемых «проблемных типов», которые уходят корнями частично в наследие языка C, а полностью – в желание дать программистам возможность обойти чрезмерно дорогую инициализацию.

Все POD-типы⁶ (3.9/10) (включая обычные `bool` и `int`) могут принести проблемы в силу общего для всех них свойства: по умолчанию они имеют неопределенное (indeterminate) значение, и заставить их принять определенное значение можно только явно.

В качестве явной инициализации вполне подойдут:

```
T v=0;    // для скалярных типов
T v={...}; // для агрегатов
T()      // при создании временных переменных или в T v=T();
new T()  // при создании объектов в динамической памяти; внимание – скобки обязательны!
```

Во всех этих случаях объект или будет явно инициализирован, или пройдет через value-инициализацию (8.5/5), в просторечии – будет обнулен.

Главное – не забывать скобки⁶.

Запись вида `T()` делает так называемую value-инициализацию, которая вызывает конструкторы подобъектов не-POD-типов и обнуляет подобъекты POD-типов. Она появилась из простых соображений: для лю-

бого POD-типа должна быть возможность без последствий сказать `T v= T();` Чтобы это было возможно – все поля POD-типа после такой инициализации должны иметь определенное значение, что и было сделано.

В реальной жизни сами по себе POD-типы не являются источником серьезных проблем.

Как вы видите, их довольно просто проинициализировать и довольно тяжело не забыть об этом.

По-настоящему опасны основанные на них проблемные типы (и здесь язык C – не виноват).

Проблемные типы – это все классы, не являющиеся POD-классами и содержащие нестатические объекты POD-типов, которые не были инициализированы в конструкторе; а также все классы, содержащие нестатические объекты проблемных типов.

Для инициализации не-POD-класса всегда используется его конструктор (если конструктора нет – используется автоматически сгенерированный пустой конструктор).

К сожалению, для того, чтобы дать возможность избежать потенциально дорогой инициализации, при работе конструктора было принято следующее правило: любой его нестатический POD-член (или базовый POD-класс), который не был явно указан в списке инициализации, не будет инициализирован.

Грубо говоря, любой не-POD-класс, имеющий нестатические POD-члены (например, `bool`), и не инициализирующий их явно, является прямым кандидатом на проблемы⁶.

Типичный пример – это разнообразные математические классы в движках. В них часто для повышения производительности делают пустой конструктор, после чего разнообразные `float x,y,z,w;` и прочие `__m128 v[4];` перестают инициализироваться, причем – для невнимательного программиста – навсегда.

Каждый раз, используя подобный тип, вы должны помнить, что сначала нужно сделать что-то, из-за чего класс завершит внутреннюю инициализацию. Типично – вызвать ему какую-то функцию или явно присвоить ему что-то.

Объект проблемного типа невозможно проинициализировать снаружи.

Это касается всех способов использования проблемных типов: агрегация нестатического члена проблемного типа, наследование от проблемного типа, создание переменной проблемного типа без инициализатора, создание временной переменной, создание объекта в динамической памяти и т.п.

Простой пример кода, иллюстрирующий проблемный тип:

```
#include <cstdio>
```

```
// Простой POD-тип
struct small_problem
{
```

```
    bool b;
    void print(const char* s) const {
std::printf(«%s: 0x%02X\n», s, b);
    };
```

```
// Проблемный тип, полученный агрегированием POD и пустым конструктором.
```

```
// Вместо агрегации и конструктора можно было просто унаследовать класс от small_problem.
```

```
struct big_problem
{
```

```
    big_problem() {}
    bool b;
    void print(const char* s) const {
std::printf(«%s: 0x%02X\n», s, b);
    };
```

```
int main()
{
```

```
    small_problem v1; v1.print(«small noinit«);
// POD, отсутствует инициализатор
    small_problem().print(«small ()init«); //
// POD, value-инициализатор T()
    big_problem v2; v2.print(«big noinit«);
// не-POD, отсутствует инициализатор
    big_problem().print(«big ()init«); //
// не-POD, T(), не поможет
}
```

Вот как выглядит

результат работы программы:

```
small noinit: 0xCC // POD, отсутствует инициализатор
small ()init: 0x00 // POD, value-инициализация через T(), все хорошо
big noinit: 0xCC // не-POD, отсутствует инициализатор
big ()init: 0xCC // не-POD, инициализация через T() не помогла
```

Подобный ход со стороны языка может лишь вызвать у программиста иллюзию, что он все сделал правильно. Например, запись

⁶ Точные определения можно прочесть в стандарте (8.5/5, 8.5/9, 12.6.2/4, 5.3.4/15). Описываемая проблема связана с 12.6.2/4. Начиная с разрешения DR 35-178-304, были введены четкие правила по поводу value- и default-инициализации, которые значительно упростили жизнь, но, естественно, оставили возможность отстрелить себе ногу при желании.

⁷ Распространенное убеждение в неэффективности таких ходов, на мой взгляд, неверно. Когда вы последний раз делали C-style cast в коде, который увидят коллеги, если там можно было бы обойтись просто `static_cast`, `const_cast` или `reinterpret_cast`?

⁸ Следует помнить, что шаблонные конструкции копирования (равно как и шаблонные копирующие операторы присваивания) инстанцируются по факту использования и не заменяют свои нешаблонные аналоги (12.8/2, 12.8/9).

⁹ Если вы всерьез захотите воспользоваться этим советом, не забудьте посмотреть реализацию `boost::value_initialized<T>` и убедиться, что вы в ней все понимаете, особенно – работу с `const`. Ну а дальше – смотрите по себе.

ВНУТРЕННИЙ ПРОДУКТ

```
my_super_fast_vec4 v= my_super_fast_vec4();
на самом деле немногим лучше записи
my_super_fast_vec4 v= v; // инициализируем себя
с собой (3.3.1/1)
```

КАК БОРОТЬСЯ С ПРОБЛЕМНЫМИ ТИПАМИ?

Не допускать их.

Минимум – явно перечислить все проблемные типы, с которыми можно столкнуться в «песочнице».

В пределе – сопоставить стоимость переделки и уровень коллег и принять решение.

В конце концов, `placement new` со специфическим конструктором никто не отменял, а встретится такая необходимость от силы пару раз – там, где идет борьба за скорость загрузки и за скорость вычислений.

Грабли от мусора в памяти (при достаточном времени) вы получите тоже от силы пару раз, а вот нервов и времени потратите – однозначно больше.

Неинициализированные члены класса

Впрочем, часто проблемные типы возникают не из-за ясного взвешенного решения, а по недосмотру.

Наиболее классический пример – в объявлении класса из-за синтаксиса языка происходит явное отделение объявления от инициализации.

Это наиболее распространенная и часто очень болезненная проблема.

Человек добавляет вполне невинный `bool` в члены класса, забывает отредактировать все конструкторы и – *voilà!* – у нас проблемный класс со всеми сопутствующими спецэффектами.

У нас совсем недавно из-за такого невинного `bool` игроку не показывались размеры призов в бонусной игре. Точнее, они сначала показывались, а потом – нет. Традиционная перемежающаяся хромата, характерная для неинициализированной или испорченной памяти. В отладочном билде все, естественно, «прекрасно работало» – потому как `0xCC` вполне сходил за `true`.

Post factum, конечно, помогает

```
assert( b==true || b==false ); // это вообще bool или оптическая иллюзия?
или даже злобное
assert( !b==b );
но знали бы прикуп...
```

К сожалению, подобные подставы компиляторы не диагностируют.

Остается полагаться либо на внимательность прог-

раммиста, либо на простой терапевтический⁷ ход: использование `const T` или длиннющих записей типа `xxx_initialized<T>`.

В случае `const T` компилятор очевидным образом поможет во всех конструкторах.

Забить прописать инициализацию во всех конструкторах просто не получится.

После компиляции можно убрать `const`, если, конечно, нет шаблонных конструкторов⁸.

Если способ с `const T` никак не подходит, можно использовать какую-нибудь разновидность `explicitly_initialized<T>` или `boost::value_initialized<T>`⁹.

Это очень простые шаблоны, которые неявно преобразуют себя в ссылки на `T` (и, в большинстве контекстов «песочницы», от `T` неотличимы) плюс, в одном случае, требуют `explicit` конструирования члена, а в другом – просто автоматически конструируют его через `T()`.

Здесь дело даже не столько в том, чтобы человек написал `explicitly_initialized<bool>`, сколько в том, что в большинстве случаев из-за лени он просто не забудет пройти по всем конструкторам.

Впрочем, у `xxx_initialized<T>` (равно как и `const T`) есть одно неоспоримое преимущество – такие члены класса не забудут проинициализировать и при добавлении нового конструктора.

МАССИВЫ В СТИЛЕ C

Массивы в стиле C – опасная штука сама по себе, а в членах классов – они просто уродливы.

Особенно уродлива, конечно, их инициализация. Точнее – отсутствие таковой.

В C++ даже не существует способа создать как член класса массив константных элементов – именно потому, что инициализировать массив констант просто невозможно.

Впрочем, красивой и производительной альтернативы массивам в C++2003, к сожалению, нет.

С точки зрения безопасности – все проще: стандартные контейнеры легко заменят C-style массивы.

Везде, где производительность не высший приоритет и размер заранее неизвестен, массивы следует заменить на `std::vector`.

Там, где важно не лезть на кучу и известен точный размер массива – используйте `boost::array` (только осторожно – об этом ниже).

Если с производительностью все серьезно и есть опасность выхода за пределы массива – то это явно не случай неопытного программиста, о котором мы

тут говорим. И уж тем более я с сомнением отношусь к случаям, когда у массива как члена класса важна производительность.

Использование контейнеров вместо C-style массива позволит применять `boost::assign`.

Пусть это мелочь, но мелочь приятная.

По поводу `boost::array` – пара мелочей.

Во-первых – не забывайте его инициализировать (у него нет конструктора по умолчанию, чтобы оставить возможность написать `boost::array<int,3> arr = { 1,2,3 };`

Во-вторых – рассмотрите возможность использования велосипедного класса-аналога `std::vector`, но имеющего фиксированную `capacity` (на базе `boost::array` или просто массивом без распределения в куче).

Очень упрощенный пример (все очень тупо и прямолинейно)¹⁰:

```
#include <boost/assign.hpp>
#include <cassert>
```

// Очень простой вектор фиксированного максимального размера

// (только для демонстрации, реальный код убран для краткости)

```
template<typename T, std::size_t N>
struct fixed_vector
{
    typedef T          value_type;
    typedef T*         iterator;
    typedef const T*    const_iterator;
    typedef T&         reference;
    typedef const T&    const_reference;
    typedef std::size_t size_type;
    typedef std::ptrdiff_t difference_type;
```

```
    fixed_vector() : the_end( elems ) {} // только для демонстрации
```

```
    template<typename IterType>
    fixed_vector(IterType b, IterType e) :
    the_end(elems) // только для демонстрации
    { for( ; b!= e; ++b ) push_back(*b); }
```

```
    size_type size() const { return the_end-elems; }
    size_type capacity() const { return N; }
```

```
    const_reference operator[](size_type idx) const
    { assert( idx<=0 && idx< size() ); return elems[idx];
    }
```

¹¹ Более радикальные примеры можно найти в Win32 API, но там – мир языка С и хотя бы понятно, почему так сделано.

¹² Иногда проблема производительности действительно встает из-за необходимости возвращать большую структуру данных (например, контейнер). В этом случае типичное решение – это действительно передача по ссылке, пока в языке не наведут порядок с moving constructors и rvalue references. Впрочем, еще более типичное решение – возвращение проху-объекта.

```
void push_back( const T& value )
{ assert( size() < capacity() ); elems[size()]=value;
++the_end; }

private:
    T* the_end;
    T elems[N];
};

template<class V, std::size_t N, class V2>
inline boost::assign::list_inserter<
boost::assign_detail::call_push_back<
fixed_vector<V,N>, V >
operator+=( fixed_vector<V,N>& c, V2 v )
{
    return boost::assign::push_back( c )( v );
}

struct foo
{
    foo() : arr(boost::assign::list_of(1)(2)(3)(4)) // яв-
ная инициализация через пару итераторов
    { arr+= { 5,6,7,8; } // цепочка push_back через
list_inserter

    const fixed_vector<int,4> arr;
    fixed_vector<int,4> arr2;
};
```

Типичное применение: в коде, где `std::vector` делается на стеке и используется преимущественно из-за `push_back` и алгоритмов.

Преимущества такого аналога `std::vector` очевидны: он быстрый, легкий, безопасный и с ним могут работать все алгоритмы. Из недостатков – он медленнее, чем `boost::array` при прочих равных.

ПЛОХОЙ ДИЗАЙН API

Очередной потенциальный источник неинициализированных данных – плохая работа проектировщика.

Очень часто данные возвращаются по ссылкам или через указатели.

Не самый плохой случай – это классика жанра¹¹:

```
bool texture::get_dimensions( int& width, int&
height, int& depth ).
```

Даже если забыть о том, что «размер текстуры» сам по себе вполне может быть ADT (abstract data type), нас все равно подстерегают проблемы.

Во-первых, программист не сможет объявить размеры константными.

То есть классическая идиома `const int width= bmp.get_width();` не работает.

Во-вторых, непонятно, что будет (и будет ли) записано в `depth`, если это таки 2D-текстура.

В-третьих, непонятно, что будет вообще в переменных, если `get_dimensions` вернет `false`.

Ну и, в конце концов, такая вот запись просто уродливо смотрится:

```
int width=0, height=0, depth=0;
const bool rv= bmp.get_dimensions(width,
height, depth);
```

Все, естественно, выглядит страшнее в реальном коде с указателями.

К счастью, лечение в этом и аналогичных случаях вполне очевидно.

Следует обернуть неудачные элементы API так, чтобы позволить использовать стандартные идиомы.

Например, возвращать композит (структуру или `boost::tuple` или `boost::optional`).

Если встает вопрос производительности – значит, следует вообще убрать такой API с глаз начинающего коллеги. Поверьте, так получится дешевле¹². Программист без излишних стрессов научится писать хороший код, а у вас не будет опасного блока, который нужно непрерывно просматривать и переписывать.

ОБЪЕКТ В ПРОЦЕССЕ КОНСТРУИРОВАНИЯ

Довольно редко (но метко) проблемы всплывают, когда идет работа с находящимся в процессе конструирования объектом. Хорошо, если подобъекты, в которые лезут, достаточно сложные, чтобы свалиться по `assert`¹³, как здесь:

```
struct foo
{
    foo() : s1(s2) {} // в реальной жизни – че-
рез несколько уровней вызовов функций
    std::string s1, s2;
};
```

Хуже, если это молчаливые PODы, или ссылки, или программисты начинают использовать `this`. (Впрочем, gcc вполне неплохо отлавливает подобные ошибки.)

Ужасает, конечно же, отсутствие диагностики:

```
struct foo
```

```
{
    foo() : r1(r2), r2(r1)
    { r1= r2; } // access violation
    int &r1, &r2;
};
```

Напомню, что речь идет о неинициализированных данных.

Разумный стандарт кодирования должен четко напомнить программистам порядок инициализации подобъектов¹⁴ и запретить использовать в конструкторах еще неинициализированные подобъекты, `this` и виртуальные функции. В большинстве случаев я бы запретил и виртуальное наследование.

Часто проблемы возникают и из-за того, что члены класса не инициализируются в списке инициализаторов конструктора. Вместо этого им присваиваются значения прямо в теле конструктора.

Это плохой стиль: вы теряете в ясности (список инициализаторов дает лучшее представление о том, что же инициализируется), во внимательности (легко пропустить какой-нибудь подобъект, требующий инициализации), в безопасности (можно случайно забыть о том, что какой-то подобъект еще не проинициализирован, и начать его использовать) и в эффективности (вынуждены выполнять присваивание вместо инициализации).

CHECKLIST

Таким образом, мы пробежались по всем основным источникам неинициализированных данных.

Технически, данные с неопределенным значением могут возникнуть только в следующих случаях:

- как значение POD-члена не-POD-класса из-за его пропуска в конструкторе (12.6.2/4),
- как значение локальной нестатической переменной POD-типа в функции (8.5/9),
- как значение POD-типа или проблемного типа выделенного по `new` (5.3.4/15),
- как значение указателя, который был передан в `delete/delete[]` и с тех пор не был изменен (5.3.5/4),
- как неинициализированный (singular) итератор (24.1/5).

Доступ к подобъектам, конструирование которых еще не начато, – `undefined behavior` (12.7/1).

Инициализация переменной своим собственным значением (`int x= x;`) – `undefined behavior` (4.1/1).

Понятия, которые имеет смысл знать (они пригодятся): POD-тип, POD-класс, `value`-инициализация.

Все `guidelines`, которые возникли по ходу – сразу под заголовком «неинициализированные данные».

¹³ Использование объекта, чье конструирование еще не начато, вызывает неопределенное поведение (`undefined behavior`) (12.7/1).

¹⁴ При конструировании объекта без виртуального наследования, сначала полностью конструируются базовые подобъекты в порядке их перечисления в списке базовых классов, и затем конструируются нестатические подобъекты класса в порядке их перечисления в списке членов класса. Порядок перечисления подобъектов в списке инициализаторов конструктора игнорируется. (12.6.2/5)



АЛЕКСЕЙ БОРЕСКОВ

>> ВНУТРЕННИЙ ПРОДУКТ

ПРОГРАММИРОВАНИЕ СОВРЕМЕННЫХ GPU

С ИСПОЛЬЗОВАНИЕМ ШЕЙДЕРНЫХ ЯЗЫКОВ ВЫСОКОГО УРОВНЯ (GLSL И CG)

Алексей Борецков – к.ф.-м.н., научный сотрудник факультета ВМиК МГУ им. Ломоносова. Автор 7 книг по компьютерной графике, две последние из которых «Расширения OpenGL» и «Разработка и отладка шейдеров». Вопросы, пожелания и предложения отправляйте ему на steps3d@narod.ru или steps3d@animekazan.net

ЗА ПОСЛЕДНИЕ НЕСКОЛЬКО ЛЕТ МЫ НАБЛЮДАЕМ СРЕМИТЕЛЬНЫЙ РОСТ ВОЗМОЖНОСТЕЙ СОВРЕМЕННЫХ GPU, ПРИ ЭТОМ ДАЖЕ САМ ТЕРМИН GPU (GRAPHICS PROCESSING UNIT, ПО АНАЛОГИИ С CPU) ПОЯВИЛСЯ СРАВНИТЕЛЬНО НЕДАВНО.

Интересно посмотреть на эволюцию графических ускорителей (так они первоначально назывались) до современных GeForce 7900 и Radeon X1900. В следующей таблице приводятся основные поколения GPU по (4).

Таблица 1. Основные этапы развития графических ускорителей

Поколение GPU	Год	Представители
Первое	1998-1999	RIVA TNT, TNT2, Rage
Второе	1999-2000	GeForce256, GeForce 2, Radeon 7500
Третье	2001-2002	GeForce 3, GeForce 4Ti, Radeon 8500
Четвертое	2002-	GeForce FX, Radeon 9700

Первые графические ускорители реализовывали так называемый фиксированный графический конвейер (rendering pipeline) (рис. 1), и все возможности программирования сводились просто к переключениям и заданиям параметров в пределах фиксированного конвейера, к растеризации уже преобразованных (CPU) примитивов, наложению на них одной или двух текстур и удалении невидимых поверхностей при помощи z-буфера.



Рисунок 1.
Фиксированный
конвейер
рендеринга

Реализовать на таких ускорителях попиксельное освещение (такого типа как в играх Doom III и Quake IV) было практически невозможно. Хотя один из нынешних сотрудников компании NVIDIA Касс Зверитт в своей дипломной работе показал, как можно реализовать

простейшее попиксельное освещение на стандартном OpenGL, но это требовало несколько десятков проходов рендеринга и было практически непригодно для приложений реального времени.

GPU второго поколения поддерживали уже обработку вершин на GPU (так называемое T&L) и расширяли возможности фиксированного конвейера.

Третье поколение GPU характеризовалось уже возможностью явного задания обработки вершин при помощи задаваемых программистом программ.

Четвертое поколение дало возможность явно задавать обработку каждого фрагмента при помощи задаваемых пользователем программ.

Таким образом, развитие графических ускорителей шло как в сторону расширения возможностей фиксированного конвейера (например, register combiners на GeForce 256, позволившие реализовать простейшие случаи попиксельного освещения за один проход), так и добавления возможностей непосредственного программирования.

Так, расширения OpenGL NV_vertex_program и ARB_vertex_program дали возможность заменить обработку вершин (преобразование и вычисление попершинного освещения) на явно задаваемую пользователем программу.

Эта программа писалась на специальном ассемблере, заметно отличающемся от ассемблера для процессоров семейства x86.

Все регистры в этом ассемблере являются 4-мерными векторами с вещественными (float) компонентами. За счет этого за одну команду можно было выполнить до 8 операций с плавающей точкой.

С другой стороны, условные конструкции в этом ассемблере практически отсутствуют – нет никаких переходов (ни условных, ни обычных), нет вызовов подпрограмм. Фактически все возможности создания условных конструкций сводились к наличию команд, вычисляющих такие функции, как нахождение покомпонентного максимума/минимума для двух векторов, покомпонентно-му сравнению двух векторов и тому подобное.

В листинге 1 приводится пример простейшей вершинной программы, соответствующей спецификации ARB_vertex_program.

Следующим, вполне естественным шагом, было добавление аналогичной возможности для обработки отдельных фрагментов, по-

лучающихся в результате растеризации графических примитивов.

Эта возможность дается в OpenGL через расширения ARB_fragment_program. Используемый для программирования обработки отдельных фрагментов язык ассемблера очень похож на язык из расширения ARB_vertex_program.

Основные отличия заключаются в наличии специальных команд для работы с текстурами и возможности условного отбрасывания фрагмента.

В результате фиксированный конвейер рендеринга в современных GPU заменен полностью программируемым конвейером (рис 2).

Graphics Hardware» (Kekoa Proudfoot, William R. Mark, Svetoslav Tzvetkov, Pat Hanrahan), представленная на конференции SIGGRAPH 2001.

Однако первым шейдерным языком высокого уровня, получившим широкое распространение, стал язык Cg (C for graphics) компании NVIDIA.

Данный язык многое взял как из языка C, так и из RenderMan Shading Language (ставшего фактически стандартом в профессиональной графике).

Как и соответствующие ассемблерные языки, язык Cg поддерживает работу с векторами (2-, 3- и 4-мерными), также поддерживается работа с матрицами.

Данный язык поддерживает все основные ус-

используемый GPU накладывает определенные ограничения на возможности, доступные программе (в результате чего не все программы на Cg могут быть выполнены для некоторых GPU).

Важной особенностью как Cg, так и других шейдерных языков является специальные модификаторы переменных, используемых при передаче данных между основной программой и шейдерами (а также и для обмена данными между вершинным и фрагментным шейдером для языка GLSL).

Помимо набора атрибутов для каждой вершины (здесь происходит обобщение традиционного графического конвейера, где набор атрибутов в вершинах предопределен; для программируемого конвейера можно связать с вершиной произвольный набор различных атрибутов) существуют также так называемые uniform-переменные.

В качестве uniform-переменных используются величины, значения которых не изменяются на протяжении рендеринга примитива (например стандартные матрицы OpenGL). Обычно при помощи uniform-переменных в шейдер передаются текстуры и глобальные параметры для шейдера. При этом такие переменные описываются с использованием модификатора uniform.

Вершинная программа обычно возвращает некоторый набор значений (в Cg, как правило, представляемый в виде некоторой структуры), которые при растеризации примитива билинейно интерполируются на весь примитив. Фрагментный шейдер получает на вход уже интерполированные значения для текущего фрагмента.

Поскольку во многих случаях гораздо удобнее использовать стандартный набор атрибутов (которые можно передавать при помощи таких функций как glVertex, glColor и тому подобных), то в Cg существует возможность в описании входных и выходных параметров указать, какому стандартному атрибуту соответствует данный атрибут.

```
struct VertexData
{
    float4 pos : POSITION;
    float3 n : NORMAL;
};
```

Так, в приведенном примере параметр pos соответствует координатам вершины (задаваемыми командой glVertex), а параметр n – вектору нормали (задаваемому командой glNormal). Подобная воз-

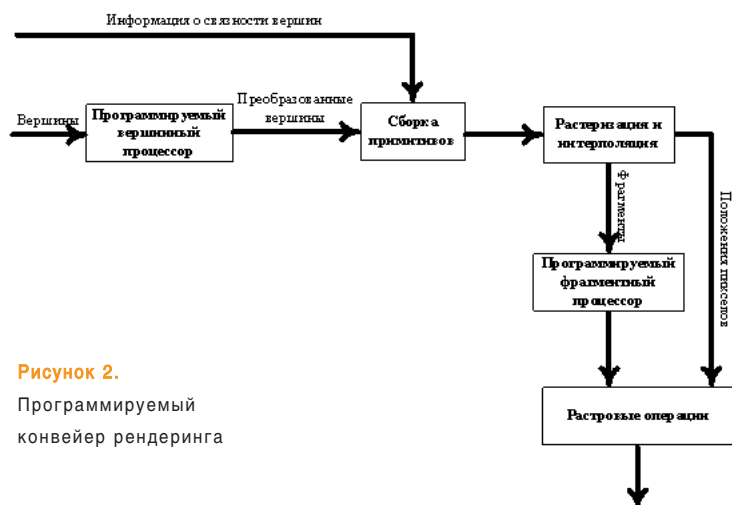


Рисунок 2.

Программируемый конвейер рендеринга

Современные GPU содержат сразу несколько вершинных процессоров и несколько фрагментных процессоров, позволяя тем самым осуществлять параллельную обработку данных (вершин и фрагментов).

Дальнейшее развитие, с одной стороны, добавляло различные виды условных команд в системы команд для вершинных и фрагментных программ (например, через расширение NV_fragment_program2), а с другой – привело к появлению и развитию высокого уровня для написания вершинных и фрагментных программ (так называемых шейдерных языков).

Одной из первых попыток создать шейдерный язык высокого уровня была работа «A Real-Time Procedural Shading System for Programmable

lowные конструкции языка C (if/else, for, while, do; не поддерживается только оператор switch), создание процедур и функций (правда, при этом не допускается рекурсия).

Есть некоторые ограничения на используемые типы – не поддерживается тип char, указатели и битовые поля. С другой стороны, есть поддержка одномерных массивов и структур.

Язык Cg предоставляет в распоряжение программиста большое количество встроенных функций, включающих в себя как стандартные математические функции (sin, cos, exp и тому подобные), так и ряд специализированных функций.

Язык Cg используется для написания вершинных и фрагментных программ, однако при этом

ВНУТРЕННИЙ ПРОДУКТ

возможность заметно облегчает использование Cg.

Язык Cg рассчитан как для работы с библиотекой OpenGL, так и Direct3D. В результате этого один и тот же шейдер на Cg может использоваться как в OpenGL-программах, так и в D3D-программах.

Еще одной интересной возможностью Cg (точнее Cg runtime) является эмулирование фрагментных программ для тех GPU, которые не поддерживают явное задание фрагментных программ (таких как GeForce 3), — в этом случае делается попытка реализовать требуемую функциональность через расширения NV_register_combiners и NV_texture_shader (однако это возможно далеко не для всех фрагментных программ).

Программы на Cg загружаются в приложение в виде исходного текста и на лету компилируются (хотя есть возможность загрузки уже откомпилированных программ).

Важным понятием для Cg (полностью отсутствующим в GLSL) является понятие профиля. Профиль определяет, в какое промежуточное представление будет компилироваться программа на Cg.

Фактически именно профиль определяет возможность успешной компиляции программы — одна и та же Cg-программа может быть успешно откомпилирована для одних профилей и не компилироваться для других (из-за отсутствия в профиле требуемых возможностей).

Несколько позже появился официальный язык для написания шейдеров в OpenGL 2.0 — OpenGL Shading Language (обычно сокращается до GLSL).

На данный момент драйверы как от компании NVIDIA, так и от компании ATI полностью поддерживают стандарт OpenGL 2.0 (то есть можно использовать все возможности OpenGL 2.0, включая поддержку GLSL при помощи механизма расширений OpenGL).

Язык GLSL также ведет свое происхождение от C и RenderMan Shading Language и во многом похож на Cg.

Одним из отличий GLSL является то, что его поддержка реализуется через драйверы, в то время как для работы с Cg необходима Cg runtime.

Еще одно отличие GLSL заключается том, что программы на GLSL компилируются сразу в код для конкретного GPU, в то время как программы на Cg сначала компилируются в промежуточный код (соответствующий данному профилю), который затем загружается через драйвер.

Также большой плюс GLSL — его «заточенность» именно под OpenGL, благодаря чему все параметры

состояния OpenGL доступны из шейдера и не требуют явной передачи из основной программы в шейдер.

Подобная возможность появилась и у последних версий Cg (но она очень ограничена, так как возможна только компиляция вершинной программы в язык ассемблера для расширения ARB_vertex_program (профиль arbvp1), а для ряда более мощных профилей она просто не поддерживается).

В языке GLSL шейдеры в явном виде не получают на вход никаких параметров и не возвращают значений. Для этого используется набор предопределенных переменных, имена которых начинаются с префикса gl_, и вводимые пользователем глобальные переменные.

Помимо модификатора uniform используется еще один модификатор varying, обозначающий переменные, которые передаются от вершинного шейдера фрагментному путем билинейной интерполяции по примитиву.

Хотя использование шейдерных языков высокого уровня и облегчило написание шейдеров, все-таки для этого лучше использовать специализированные среды (IDE). Такой средой для языка Cg является FxComposer (от компании NVIDIA), а для языка GLSL — RenderMonkey (от компании ATI).

Обе эти среды могут быть бесплатно скачаны с сайтов компаний (<http://developer.nvidia.com>, <http://www.ati.com>).

Рассмотрим использование языков Cg и GLSL на простом примере — создании шейдеров для рендеринга прозрачного объекта с отражением и преломлением (см. скриншоты).

К сожалению, точный расчет отражения и преломления для большинства объектов слишком сло-

жен для рендеринга в реальном времени, здесь гораздо лучше подходит трассировка лучей.

Поэтому обычно используется упрощенная модель — отражения и преломления считаются только для ближайшей к наблюдателю (камера) поверхности и используются для индексирования в кубическую текстурную карту для получения цветов для отражения и преломления (рис. 3).

Для получения реалистично выглядящего изображения удобно использовать одну из приближенных моделей расчета коэффициента Френеля, отвечающего за распределение энергии при отражении и преломлении.

В простейшем случае задачей вершинного шейдера является осуществление стандартного преобразования вершины при помощи произведения матриц modelview и projection, преобразование вектора нормали и вычисление вектора на наблюдателя.

Обычно для преобразования направлений служит обращенная и транспонированная верхняя левая 3x3 подматрица матрицы modelview. Добавим еще одну матрицу для преобразования векторов направления — это позволит нам независимо от объекта вращать кубическую текстурную карту.

Таким образом, вектор нормали преобразуется путем умножения сразу на две матрицы, а вектор на наблюдателя находится как разность текущей вершины и положения наблюдателя и умножается на матрицу, соответствующую повороту кубической текстурной карты.

Соответствующий код для языков Cg и GLSL приводится в листингах 2 и 3.

Задача фрагментного шейдера более сложная.

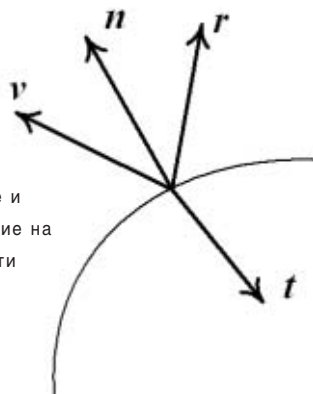
Во-первых, необходимо нормировать (то есть привести к единичной длине) полученные из вершинного шейдера векторы нормали и направления на наблюдателя, поскольку значения, поступающие в фрагментный шейдер, получаются путем билинейной интерполяции передаваемых вершинных шейдером значений, а билинейная интерполяция в общем случае изменяет длину вектора.

Во-вторых, необходимо найти отраженный и преломленный векторы, соответствующие вектору нормали и вектору на наблюдателя. Для этого удобно воспользоваться стандартными (как для Cg, так и для GLSL) функциями reflect и refract.

В-третьих, необходимо найти приближенное значение коэффициента Френеля.

Рисунок 3.

Отражение и преломление на поверхности объекта



ЛИТЕРАТУРА

1. www.steps3d.animekazan.net, www.steps3d.narod.ru
2. Расширения OpenGL, А.В. Бореков, СПб: БХВ-Петербург, 2005
3. Разработка и отладка шейдеров, А.В. Бореков, СПб:БХВ-Петербург, 2006.
4. The Cg Tutorial, Randima Fernando, Mark J. Kilgard, AddisonWesley, 2003
5. <http://developer.nvidia.com>
6. <http://www.3dlabs.com>

Заключительный шаг заключается в получении цветов из кубической текстурной карты, соответствующих отраженному и преломленному векторам, и объединении их при помощи коэффициента Френеля.

Версии фрагментного шейдера для языков Cg и GLSL приводятся в листингах 4 и 5.

Полный исходный текст (включая ряд полезных для работы с шейдерами классов) вы можете найти на нашем сайте www.gamedeveloper.ru.

Листинг 1. Пример вершинной программы на расширении ARB_vertex_program

```
!!ARBvp1.0
ATTRIB pos = vertex.position;
PARAM mvp [4] = { state.matrix.mvp };

# store texcoord
ord [0]
MOV result.texcoord [0], vertex.texcoord [0];

# copy primary
and secondary colors
MOV result.color, vertex.color;
MOV result.color.secondary, vertex.color.secondary;

# transform
position into clip space
DP4 result.position.x, vertex.position, mvp
[0];
DP4 result.position.y, vertex.position, mvp
[1];
DP4 result.position.z, vertex.position, mvp
[2];
DP4 result.position.w, vertex.position, mvp
[3];

# we're done
END
```

Листинг 2. Вершинная программа на Cg для прозрачного объекта

```
//
// Refractive glass vertex shader
//
```

```
struct VertexData
{
    float4 pos : POSITION;
    float2 texCoord : TEXCOORD0;
    float3 n : NORMAL;
};
```

```
struct FragmentData
{
    float4 pos : POSITION;
    float2 texCoord : TEXCOORD0;
    float3 n;
    float3 e;
};
```

```
);

FragmentData main ( VertexData IN,

uniform float4 eyePos,

uniform float4x4 cubeMapMatrix,

uniform float4x4 mvp, // P * M

uniform float4x4 mvi, // M^(-1)
{
    FragmentData OUT;
    float3 p = mul ( mvp, IN.pos ).xyz;
    float3 n = mul ( mvi, float4 ( IN.n, 0 ) ).xyz;

    OUT.n = mul ( float4 ( n, 0 ),
cubeMapMatrix ).xyz;
    OUT.e = mul ( float4 ( p - eyePos.xyz, 0 ),
cubeMapMatrix ).xyz;
    OUT.pos = mul ( mvp, IN.pos );
    OUT.texCoord = IN.texCoord;

    return OUT;
}
```

Листинг 3. Вершинная программа на GLSL для прозрачного объекта

```
//
// Refractive glass vertex shader
//
uniform vec4 eyePos;
uniform mat4 cubeMapMatrix;

varying vec3 n;
varying vec3 e;

void main(void)
{
    vec4 pos = gl_ModelViewMatrix * gl_Vertex;

    gl_Position = gl_ModelViewProjectionMatrix *
gl_Vertex;
    n = gl_NormalMatrix * gl_Normal *
mat3 ( cubeMapMatrix );
    e = ( ( pos - eyePos ) * cubeMapMatrix
).xyz;
}
```

Листинг 4. Фрагментная программа на Cg для прозрачного объекта

```
//
// Refractive glass fragment shader
//
```

```
struct VertexData
{
    float4 pos : POSITION;
    float2 texCoord : TEXCOORD0;
    float3 n : NORMAL;
};
```

```
struct FragmentData
{
    float4 pos : POSITION;
    float2 texCoord : TEXCOORD0;
    float3 n;
    float3 e;
};

float4 main ( FragmentData IN,
uniform samplerCUBE envMap ) :
COLOR
{
    const float eta = 0.9;

    float3 en = normalize ( IN.e );
    float3 nn = normalize ( IN.n );
    float3 r = reflect ( en, nn );
    float3 envColor = texCUBE ( envMap, r ).xyz;
    float fresnel = clamp ( abs ( dot ( en, nn ) ), 0.1,
0.9 );
    float3 t = refract ( en, nn, eta );
    float3 refractionColor = texCUBE ( envMap, t
).xyz;

    return float4 ( lerp ( envColor, refractionColor,
fresnel ),
1.0 );
}
```

Листинг 5. Фрагментная программа на GLSL для прозрачного объекта

```
//
// Refractive glass fragment shader
//

uniform samplerCube envMap;

varying vec3 n;
varying vec3 e;

void main (void)
{
    const float eta = 0.9;

    vec3 en = normalize ( e );
    vec3 nn = normalize ( n );
    vec3 r = reflect ( en, nn );

    vec3 envColor = textureCube ( envMap, r ).rgb;
    float fresnel = abs ( dot ( en, nn ) );

    fresnel = clamp ( fresnel, 0.1, 0.9 );

    vec3 t = refract ( en, nn, eta );
    vec3 refractionColor = textureCube ( envMap, t
).rgb;

    envColor = mix ( envColor, refractionColor, fres-
nel );

    gl_FragColor = vec4 ( envColor, 1.0 );
}
```



АЛЕКСАНДР ЛОГИНОВ

>> УПРАВЛЕНИЕ



НАШИ РАЗРАБОТЧИКИ ЗАРУБЕЖОМ

ЦЕЛЬ ДАННОГО МАТЕРИАЛА РАССКАЗАТЬ О ЖИЗНИ И РАБОТЕ БЫВШИХ СОТРУДНИКОВ РОССИЙСКИХ КОМПАНИЙ ЗА РУБЕЖОМ. Статья была создана на основе опроса 50 разработчиков разных уровней и специальностей, которые согласились поделиться информацией при условии гарантии анонимности, поэтому вы не найдете здесь конкретных имен и названий компаний. Мы хотим лишь показать, как различаются отношения в коллективе, условия труда и жизнь вполне конкретных людей. Чуть ниже расположены три маленькие и наиболее характерные истории от наших бывших соотечественников. Завершают повествование результаты анонимного анкетирования.

ТИП КОМПАНИИ. БОЛЬШАЯ СЕМЬЯ.

Местоположение: Чехия
Количество сотрудников: 35
Проект: PC
Время работы: 26 месяцев
Профессия: Дизайнер

Так получилось, что я был вынужден уехать на время в Прагу, где обстоятельства заставили меня задержаться сверх положенного срока. Поиск работы в смежных областях (в Москве я работал дизайнером в одной из известных игровых компаний) не дал особых результатов, пока на оставленное в Интернет резюме не пришло предложение от небольшой местной игровой студии. Так, как компания оказалась с англо говорящими сотрудниками, то я быстро прижился на новом месте, где уже работаю более двух лет. Наша студия расположена в живописном

месте в нескольких километрах от чешской столицы. Производственная база занимает небольшой особняк начала века, отреставрированный и оснащенный всем необходимым оборудованием. Кроме самого офиса, где мы работаем, в доме есть кинозал, спортзал, кухня, комната отдыха, ванная комната, а также несколько спален. Большинство сотрудников предпочитают по окончании рабочего дня возвращаться домой, но некоторые (вроде меня), не отягощенные собственным жильем или финансами предпочитают ночевать в офисе. Если наступает Crunch Mode (из моего опыта – пару раз в год), то ночевать в офисе приходится даже семейным людям. Из дополнительных бонусов можно считать бесплатные (оплачивается руководством) холодильники с едой (пицца, чипсы) и напитками (кофейные автоматы, кола, минералка). Рабочий день нормирован, но иногда приходится задерживаться на несколько часов для завершения того или иного отрезка работы, не говоря про Crunch периоды. Отношения в коллективе скорее дружеские, чем холодные. Конечно, бывают некоторые конфликты, но они обычно решаются в частном порядке. Отношения с начальством, теплые, но сразу чувствуется дистанция. То есть, в отличие от моего российского опыта, руководство не пьет вместе с рядовыми сотрудниками пиво и уж тем более не пытается заигрывать мифическими бонусами («вот вы работайте, а как продадим игру все заживем, как шейхи и короли»). У тебя есть задача и контракт – работай. Кроме того, каждый конкретный сотрудник знает, за что отвечает и не лезет в дела другого. Здесь также не

встретишь того, что мне доводилось видеть в некоторых отечественных компаниях, когда программист пытается научить игрового дизайнера правильно позиционировать игру. Если говорить о материальном вопросе, то зарплата здесь не слишком отличается от той суммы, что мне приходилось слышать от московских коллег, но при этом она официальная. Доволен ли я работой? Да. Не хватает только более теплого и доверительного отношения и русской речи.

ТИП КОМПАНИИ. WANNABE-КОРПОРАЦИЯ.

Местонахождение: Бельгия
Количество сотрудников: 120
Проект: Xbox 360, PS3
Время работы: 16 месяцев
Профессия: Программист

После завершения проекта в одной из отечественных студий я получил заманчивое предложение от своих западных друзей и, получив рабочую визу, переехал в Бельгию для работы над так и не анонсированным проектом. Моя компания оказалась с игровым ветераном с большими западными инвестициями, расположенная в центре города. Мне четко объяснили мои обязанности, показали штатной расписание с обязательным посещением нескольких курсов. Как мне сообщили в первый рабочий день, мне рекомендуется пройти языковой курс (если я планирую получить гражданство), а также несколько специальных курсов по повышению моей технической квалификации, необходимые для моей работы со специальными приложениями. Кроме всего прочего, я мог выбрать из внушительного

списка еще несколько курсов на свой вкус. Все занятия, также как и жилье, недалеко от работы, оплачивало руководство студии. Рабочий процесс в отличие оттого, что мне пришлось пережить в России, отличался четкостью задач. Мой начальник требует еженедельной отчетности, но при этом не вмешивается в процесс. Впрочем, были и некоторые нюансы, которые мне сразу не понравились:

1. Разобщенность коллектива. Все занимаются своим делом. Общаются только по деловым вопросам. Редко кто дружит и общается вне стен офиса.

2. Развитая система стукачества. Стучат по поводу и без повода. Причем, это поощряется самим начальством. Если в Российской компании один программист мог прикрыть другого или дизайнер моделилера, то здесь каждый за себя. Если ты допустил ошибку, то об этом немедленно сообщает твои ближайшие коллеги из самых лучших (спасти проект и подняться в лице начальства) побуждений.

3. Ограниченный рабочий день. По закону мы не можем работать больше, чем это установлено местным КЗОТом, так, что если прогудел гудок, то надо все бросать и идти домой. То, что в России можно было доделать в дополнительное время – здесь должно быть сделано в указанный в таймлайне срок.

Немного о внутренней обстановке. Во всех помещениях стоят автоматы с минеральной водой, бесплатные кофейные автоматы. Есть комната отдыха с приставками, но здесь практически никто не играет. Оставаться в студии на ночь строго запрещено. Компания обязана позаботиться о твоём жилье заранее. И, наконец, самый важный момент – оплата. Здесь она выше, чем в России, но значительно ниже, чем в США. Переработка оплачивается, но руководство старается ее избегать. А так все относительно. Сложно сказать, где лучше. В России лучше отношения между людьми, но здесь усовершенствован рабочий процесс и высокая заработная плата.

ТИП КОМПАНИИ. ИГРОВОЙ КОНВЕЙЕР.

Местонахождение: США

Количество сотрудников: 500

Проект: PC, Xbox 360, PS3

Время Работы: 8 месяцев

Профессия: Художник

Работать в крупной американской компании мне всегда казалось модно и престижно, особенно если

тебе предлагают не плохую зарплату и возможность получить опыт в кругу ветеранов игровой индустрии. Моя мечта закончилась для меня печально – я вернулся обратно на Родину. Оказалось, что в большой компании ты фактически являешься звеном в большом механизме, которому платят минимальные деньги. За время моей работы над проектом, у нас в студии сменилось до 45 процентов состава. Причем, отбоя в желающих никогда не было – студенты и выпускники местных технических университетов с удовольствием работают ради получения опыта. Как вытекающее отсюда отношение к рядовым сотрудникам, которых очень часто увольняют после проекта. Все, что вы слышали от разгневанной жены (знаменитый тред на Live Journal о переработках в EA) программиста в EA полная, правда. За время работы мне часто приходилось оставаться в дополнительное, неоплачиваемое время. Правда, нам уровень сервиса внутри компании был достаточно высок (бесплатные напитки, базовая еда, спорт зал, комната отдыха, путешествия за счет фирмы по окончании проекта), но это радовало только первое время. Кроме фактической работы на износ я постоянно ощущал давление со стороны своего менеджера. Это выражалось в замечаниях, придирках и попытке сломать меня как личность. Более того, как часть системы я имел лишь частичное представление о том продукте, что мы разрабатывали. В результате всех этих факторов у меня началась депрессия, и я предпочел сменить место работы, присоединившись к небольшому стартапу, который я в последствии покинул по личным обстоятельствам. Сейчас я работаю в смежной индустрии в одной из отечественных компаний и чувствую себя вполне довольным.

ОПРОС

1. Довольны ли вы своей работой в западной компании?

- 35% – Доволен
- 25% – Скорее да, чем нет
- 16% – Не доволен, но вынужден смириться
- 12% – Недоволен и планирую сменить работу
- 11% – Не знаю

2. Насколько условия труда в западной компании отличаются в положительную сторону от российской?

- 45% – Сильно отличаются в лучшую сторону
- 18% – Не отличаются
- 17% – Затрудняюсь ответить
- 20% – Отличаются в худшую сторону

3. Насколько уровень заработка в западной компании отличается от российской?

- 79% – Значительно больше
- 11% – Также
- 7% – Затрудняюсь ответить
- 3% – Меньше

4. Насколько отношения в коллективе в западной компании отличаются от российской?

- 36% – Лучше
- 25% – Также
- 12% – Затрудняюсь ответить
- 27% – Хуже

5. Планируете ли вы вернуться в русский гейм дев?

- 33% – Да
- 18% – Думаю над этим
- 37% – Нет, но если что-то изменится
- 12% – Нет, останусь здесь жить.

6. Смогут ли российские компании догнать по уровню производства своих западных коллег?

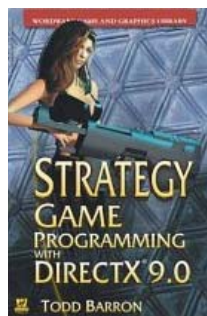
- 29% – Уже догнали
- 17% – Вряд ли, но близко
- 38% – Нет, никогда
- 16% – Когда их купят западные издатели

В следующем номере мы поговорим о впечатлениях иностранных разработчиков от наших игровых студий.



АЛЕКСАНДР КАЛИНИН

>> ЗА СЕМЬЮ ПЕЧАТЯМИ



«Shaders for Game Programming and Artists»

АВТОРЫ	Себастьян Лаурент (Sebastien Laurent)
ИЗДАТЕЛЬСТВО	Thomson Course Technology
ГОД ВЫХОДА ISBN	2004 1-59200-092-4

НИ ОДНА КНИГА ИЗДАТЕЛЬСТВА «THOMSON COURSE TECHNOLOGY» ДО ВЫХОДА «SHADERS FOR GAME PROGRAMMING AND ARTISTS» не была посвящена шейдерам. Редактор серии с милым сердцу названием «Game Development», Андре Ламот, сам автор блестящих книг, был прекрасно осведомлен о столь вопиющем пробеле в ассортименте. Однако он не спешил: найти человека, который смог бы написать книгу, полезную начинающим и дающую исчерпывающее введение в предмет, очень сложно. Но, как показала практика, возможно!

Книга написана на редкость удачно – нет ни одной банальной или неинтересной главы. Во введении в меру кратко, при этом достаточно для получения твердых первоначальных представлений приведен обзор истории шейдерной технологии, основ трехмерной графики и архитектуры шейдеров. Для их разработки использовалась RenderMonkey 1.5 (далее RM) от nVidia, базовые навыки работы с которой освещались в третьей главе. А в следующей, на основе полученных знаний, читатели уже смогли написать свои первые, простейшие шейдеры – вывод объекта на экран, текстурирование и визуализацию объекта вместе с копией. На этом первая часть книги закончилась, и наступило время для более интересных вещей. Вторая часть освещает технику использования предварительно прорендеренных текстур. В третьей части «Shaders...» внимание концентрируется на повышении реалистичности картинки: обсуждаются фундаментальные темы – материалы и освещение. У разобравшихся с этой частью в дальнейшем не должно возникнуть принципиальных сложностей с освоением более продвинутых техник и методов. Заключительная, четвертая часть книги, посвящена созданию более продвинутых шейдеров и реализации эффектов, которые было сложно отнести к ранее рассмотренным. Чего здесь только нет: туман, скелетная анимация, анимация по методу ключевых кадров, создание рельефных карт, теней и разработка механизма изменения уровня детализации.

ПОЛЬЗА

Конечно, книга и сама чрезвычайно полезна. Кроме того там можно найти:

- краткое, но вполне достаточное описание HLSL;
- подробный рассказ о работе в RM;
- начиная со второй части, в конце каждой главы вы найдете 1-2 упражнения, из которых почти все представляя собой самостоятельную ценность. Ответы с комментариями прилагаются. Кроме того, полный перечень всех основных, часто используемых шейдеров, находится в конце книги

ОТЛИЧИЯ ОТ ДРУГИХ В ДАННОЙ ТЕМАТИКЕ

Почти полное отсутствие необходимых предварительных знаний (но не мозгов!)

- отсутствие привязки к конкретному графическому API
- исключительная логическая стройность и доступность изложения

КОМУ НУЖНА

Если вы не слишком уверенно чувствуете себя в шейдерах – то эта книга для вас. Если вы вообще никогда не занимались ими, то выбор любой другой книги – большая ошибка. Идеально подойдет художникам, желающим самостоятельно, без помощи программистов, реализовывать собственные эффекты.

КОМУ НЕ НУЖНА

Если вы «seasoned professional», то ничего нового, скорее всего, для себя не найдете. Однако и в этом случае просмотреть книгу будет нелишним.

ОБЩИЙ ВЫВОД

«Shaders...» – лучшая книга для первоначального знакомства с шейдерами. Несмотря на это, полученные знания и навыки можно с успехом повсеместно применять в реальной жизни. Там нет воды и банальных, кочующих из одного учебного пособия в другое «фундаментальных», никому не нужных истин. Книга содержит свежую, своевременную и интересную информацию. Советую ее буквально всем.



«Проектирование и архитектура игр»

АВТОРЫ	Эндрю Роллингз, Дэйв Моррис
ИЗДАТЕЛЬСТВО	Вильямс
ГОД ВЫХОДА	конец 2005 (русскоязычное издание)
ISBN	5-8459-0914-7

СРАЗУ НУЖНО ОГОВОРИТЬСЯ – ПОД «ПРОЕКТИРОВАНИЕМ» ИГР В ЭТОЙ КНИГЕ АВТОРЫ ПОДРАЗУМЕВАЛИ НЕ ТОЛЬКО ЛИШЬ ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ (разработка архитектуры, написание тех. заданий и т.д.). Что же? Лучшим ответом на этот вопрос послужит сама книга. Целиком.

В целом «Проектирование и архитектура игр» делится на три большие части – проектирование игр, управление рабочей группой проекта и собственно разработка архитектуры. Впрочем, эти сведения с легкостью можно найти и в оглавлении. После первого прочтения «по диагонали» – а людям заинтересованным совершенно точно от нее будет трудно оторваться – на вопрос «о чем эта книга?» может возникнуть только один ответ «обо всем». И только за последующим внимательным и неторопливым изучением в памяти откладываются масса пропущенных важных деталей и приходит уверенность, что книга действительно «о проектировании игр». Среди затронутых вопросов отмечу:

- разбор процесса выработки концепции игры и ее дальнейшего развития;
 - сущность эскизного проекта и подходов к его созданию;
 - мнение авторов по поводу анатомического строения геймплея и возможностей его проектирования;
 - организацию документации;
 - обзор типов игрового баланса и подходов и их обеспечению;
 - вопросы управления коллективом – от распределения ролей до улучшения морального климата;
 - конвейерное программирование – достоинства и недостатки;
 - система контрольных точек, ее организация и вариации;
 - влияние на них технологий – целесообразность и эффективность;
- И многое-многое другое...

В целом осталось впечатление как об этой «энциклопедии дела». Конечно, большинство советов и рекомендаций из разряда «просто, но не очевидно», но не все. Шаблоны проектирования и варианты их программной реализации, разбор эффективных моделей организации производственного цикла, подробный рассказ о натуральных макетах и компонентах, стандарты организацией проектной документации, обзор типичных ошибок в немыслимом числе сопутствующих областей – все это будет интересно даже весьма компетентному читателю.

ПОЛЬЗА

Огромное количество примеров иллюстрирующих буквально каждый аспект книги.
- весьма приятный словарь терминов.
- замечательный пример проектной документации (пожалуй лучший из тех, что я видел в книгах) – игра Balls!

ОТЛИЧИЯ ОТ ДРУГИХ В ДАННОЙ ТЕМАТИКЕ

Крайне редко можно встретить книгу большого объема, в которой практически не было бы значительного количества справочных сведений, напрямую перепечатанных из общеизвестных источников.
В этой книге нет воды. Все по делу.

КОМУ НУЖНА

Осмелюсь высказать предположение, что эта книга будет полезна буквально всем, причастным к игровой индустрии. Даже если вам кажется, что ничего нового из вышеупомянутых вопросов вам не откроется, все равно советую вам купить эту книгу – тем более, что сейчас она легко доступна.

КОМУ НЕ НУЖНА

Всем тем, кто не относится с вышеупомянутой категории.

ОБЩИЙ ВЫВОД

Первое английское издание «Проектирования...» не только стало классическим геймдев-произведением, но и убедительно продемонстрировало компетентность авторов. Большинство из высказанных ими прогнозов касательно развития тех или иных аспектов оправдались. Беспрецедентный охват материала. Огромное количество наблюдений и фактического материала. И еще один веский, но весьма неприятный аргумент. Лучшее из того, что есть на русском языке. Потому что единственное. По состоянию на текущий момент.



> ДОРОЖЕ ТЫСЯЧИ СЛОВ



OUTPOST KALOKI X (XBOX 360)



АРТ-ДИРЕКТОР: БРЕНТ ФОКС
 КОНЦЕПТ/2D-АРТ: ШОН БОЙЛЗ
 3D-МОДЕЛЛЕР: ДЖЕФФ ВИГГИНС

ХОТИТЕ СТАТЬ НАШИМ ПАРТНЕРОМ?

Соболев Максим sobolev@gameland.ru
Менеджер по рекламе

С ЧЕГО НАЧИНАЮТСЯ ИГРЫ



gamedeveloper (game)land
ВЕДУЩИЙ ЖУРНАЛ ОБ ИГРОВОЙ ИНДУСТРИИ РОССИЯ

> LOG ON: www.gamedeveloper.ru

Получи два первых номера бесплатно