DirectX и Delphi
искусство программирования

МИХАИЛ ФЛЕНОВ

Реалистичные графические эффекты
Пиксельные и вершинные шейдеры
Оптимизация графики
2D- и 3D-эффекты
Современные графические технологии

+CD

УДК 681.3.06
ББК 32.973.26-018.1
Ф69

Фленов М. Е.

Ф69 DirectX и Delphi. Искусство программирования. — СПб.: БХВ-Петербург, 2016. — 384 с.: ил.

ISBN 978-5-94157-870-2

Рассмотрено использование популярной библиотеки DirectX при программировании графических эффектов в Delphi. Подробно описано применение основных методов и интерфейсов DirectX. Большое внимание уделено технологии использования вершинных и пиксельных шейдеров для создания реалистичных изображений. Показано, как эффективно программировать огонь, электрические разряды, зеркала и другие визуальные эффекты, используемые при разработке демонстрационных роликов (Demoscene). Компакт-диск, прилагаемый к книге, содержит листинги примеров из книги и дополнительную информацию по DirectX.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Юрий Рожко*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Инны Тачиной*

Подписано в печать 28.03.06.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 30,96.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-94157-870-2

Оглавление

ГЛАВА 1

Введение в Demo и DirectX

Прежде чем окунуться в великолепный мир программирования, давайте немного поговорим о демо-сцене и DirectX. Познакомившись с историей Demo, мы постараемся ощутить дух развития всей сцены. История сцены достаточно интересна и я рекомендую прочитать ее в любом случае.

Далее нас ждет введение и теория DirectX. Вот эту часть можно пропустить, если вы уже имеете опыт программирования графики в Windows. В данной книге под графикой мы будем понимать именно DirectX, а не GDI (Graphic Device Interface, интерфейс графических устройств), если явно не указана используемая графическая технология.

1.1. История демо-сцены

Первые ролики для платформы PC содержали только видеоэффекты и не всегда имели звуковое сопровождение, потому что это были 80-е годы, и не каждый PC-совместимый компьютер имел звуковую карту. В те времена производители устанавливали только один маленький динамик PC Speaker, возможности которого ограничивались писком. Но впоследствии звук стал неотъемлемой частью демо, и программисты умудрялись создать шедевры даже на пищалке PC Speaker, а когда звуковая карта стала устанавливаться практически в каждый компьютер, качество звука значительно повысилось.

Здесь меня могут упрекнуть в искажении фактов, ведь даже в 80-е годы были компьютеры, которые могли воспроизводить достаточно качественный по сравнению с PC Speaker звук. Да, были платформы типа Amiga, которые также оказали на демо-сцену серьезное влияние, но мы рассматриваем именно PC-платформу, а она не была предназначена для игр и графики, и изначально здесь все ограничивалось ASCII-графикой и примитивным звуком.

Первые ролики создавали в основном крэкеры (взломщики программ) или профессиональные программисты (хакеры). Крэкеры создавали небольшие ролики, которые содержали информацию о самом крэкере или группе. Такие демо-ролики вставлялись во взломанные программы, и любой пользователь мог увидеть, благодаря кому программа стала бесплатной. Эти демо-ролики были максимально простыми, потому что основным их требованием было — минимальное количество места при максимальном эффекте (производительности и красоты). Именно крэкеров считают основателями культуры демосцены, по крайней мере на платформе PC.

Программисты расширили эту идею и стали создавать более сложные деморолики, которые стали существовать как отдельные программы. Цель таких роликов — поразить пользователя и показать невиданные возможности простого PC-совместимого компьютера.

В середине 90-х я увидел демо-ролик Second Reality от группы Future Crew, который был создан в 1993 году (рис. 1.1). После его просмотра у меня осталось не передаваемое словами ощущение восторга. На 386-м компьютере с частотой 40 МГц я смог увидеть действительно впечатляющие эффекты и великолепный звук. Это уже был целый видеоклип, в котором графика великолепно гармонировала со звуком, а главное — я поразился, как такой слабенький компьютер может рассчитывать такие сложные сцены в реальном времени.

Рис. 1.1. Скриншот ролика Second Reality

Я не могу сказать, что именно группа Future Crew стала новатором в новом поколении демо, но для меня и многих других именно эта "дема" перевернула жизнь. С тех пор я увлекся графикой и графическими эффектами и с тех времен стал усиленно интересоваться этой культурой. Теперь демо-ролик — это не просто шанс программисту показать свои способности, но и продемонстрировать другим новые, невиданные возможности компьютера. Ролик Second Reality сочетал в себе умопомрачительную по тем временам 2D- и 3D-графику в сочетании с великолепной музыкой, и при этом не использовалось никаких графических ускорителей.

Период с 1993-го по 1995-й дал демо-сцене новый виток гонки за 3D-эффектами. Появление процессора Pentium позволило усложнить графику и создавать новые эффекты, которые рассчитывались в реальном времени, но при этом еще не использовалось никаких ускорителей, без которых мы сейчас не можем представить себе ни одну графическую игру или даже графическую программу.

Да, разрешение экрана в демо-роликах было не таким высоким и благодаря этому экономилось большое количество процессорного времени. И все же, для создания таких эффектов требовались немалые усилия, знания и умения программистов. Даже при небольшом разрешении ролики смотрелись очень эффектно, потому что в те времена стандартом был монитор 14 дюймов, а о 15 или 17 дюймов приходилось только мечтать, потому что стоили они очень много. Это сейчас мы сидим за 17-дюймовыми "трубами" или панелями и тут без графического ускорителя не обойтись, иначе при разрешении 320×200 точек на экране будет изображение из больших квадратов, а не точек.

Так уж повелось в нашей жизни, что программисты в большинстве случаев не умеют рисовать (у меня тоже проблемы с художеством), поэтому в создание демо-сцен стали включаться художники и музыканты. Теперь, для того чтобы поразить пользователя и тем более победить на конкурсе, необходима была полная гармония графических эффектов, текстур, звука и видеоряда. Именно поэтому над созданием роликов стали трудиться целые команды.

С увеличением мощности компьютеров усложнялись и демо-ролики. Этому способствовало и появление новых графических технологий, упрощающих программирование, таких как OpenGL и DirectX, а также появление графических ускорителей. Демо-ролики уже перестали быть двумерными и все больше покоряли третье измерение.

Некоторые программисты совершенствовались и шли в ногу со временем, а некоторые противились этому. Таким образом, демо-сцена стала разделяться на несколько основных направлений:

- классические демо-ролики, размер исполняемого файла которых не превышает 4 или 64 Кбайт и при этом не используется никаких графических технологий и ускорителей. В таких демо для создания сцены может использоваться только математика, и все расчеты должны производиться в реальном времени;
- 2D, 3D или смешанная демо-сцена;
- Demo с использованием графического ускорителя и технологии OpenGL;
- Demo с использованием графического ускорителя и технологии DirectX.

Самое сложное, на мой взгляд, это первое. Создать демо-ролик с хорошим эффектом в 4 Кбайт очень непросто. Конечно, задача усложняется, если ис-

пользовать Windows, но и в MS-DOS эта задача не такая уж простая. И все же, виртуозы своего дела умудряются и на такие шедевры. Например, на сайте **http://www.scene.org** можно найти "демку" fr026, точнее сказать, это графический эффект. Размер исполняемого COM-файла — 34 байта, а исходный код на языке ассемблера умещается на одной странице (листинг 1.1).

Листинг 1.1. Исходный код "демки" fr026

```
; fr-026: 34b mul+cycle (uc 6.22 size optimizing compo entry)
; 1st place
; code: ryg (original concept) & kb (additional opcode crunching)
;
; rules were to write a "colorcycling" (no real colorcycling was
; required, only colorcycling-like animation) effect that display
; a x * y-ish pattern somewhere on the screen. the palette was
; required to look like the original version.

org  256

  mov  al, 13h  ; ...load al with 13h (mode number)
  int  10h      ; set mode 13h

  mov  bp, 320  ; screen is 320 pixels wide

x les  ax, [bx] ; es = end of mem (9fffh), ax = 20cdh
  cwd           ; => dx = 0 (for idiv)
  mov  ax, di   ; get dest address
  idiv bp       ; div by width => ax=y coord, dx=x coord
  imul dx       ; mul x by y
  stosb         ; store color in vram
  mov  dx, 3c9h ; dac data
  bt   cx, 6    ; bit 6 of cx => carry flag
  salc          ; set al from carry flag
  xor  ax, cx   ;=> all this is functionally equivalent to something like
                ; "test al, 64" / "jz skip" / "not al" / "skip: blah"
                ; which should be somewhat easier understandable,
                ; but bigger.
  out  dx, al   ; write grayscale pixel values
  out  dx, al
  out  dx, al
  loop x        ; pixel loop.
  loop x        ; skip cx = 0 to get the cycling effect!
```

```
; that's all there is to it.
; this is the 7th (or 8th) version; the first try was
; 49 bytes, from then on we successively made it smaller.
```

В листинге код эффекта показан как есть, но если из него убрать все комментарии, то его размер уменьшится почти в два раза.

Отдельной веткой шло развитие Amigo и ZX Spectrum демо-сцены, но мы эту тему опускаем в связи с досрочной смертью платформы, поэтому данное направление не рассматриваем.

Создание роликов с использованием графического ускорителя также могло разделяться на несколько направлений, в которых могло быть ограничение на размер исполняемого файла или не было этого ограничения.

В настоящее время разновидностей демо-сцены намного больше, здесь и Flash-демо, и создание видеороликов в графических пакетах, таких как 3ds Max, цифровые картинки, полностью созданные на компьютере, или даже просто цифровая музыка. Описать процесс создания всех разновидностей демо-сцены невозможно, потому что они безграничны. Можно ограничиться только теми направлениями, которые стали стандартом для показа на демовечеринках и конкурсах, но даже для рассмотрения этой темы нужна книга размером с "Войну и мир". Именно поэтому я ограничился только демороликами с использованием ускорителей и технологии DirectX, да и как я уже сказал, из меня художник и музыкант не получился, и для изучения этой стороны демо-сцены лучше почитать специализированную литературу.

Может показаться, что демо-сцена — это всего лишь видеоклип на какую-то тему и под определенную музыку. Если под словом "клип" понимать то, что мы видим на телеканале MTV, то к демо-сцене можно отнести единичные творения. Из российских клипов к сцене с натяжкой относятся клипы Глюкозы. Аниматоры тут серьезно постарались и работы получаются законченными и интересными, а по поводу стиля музыки — это дело вкуса, и в демосцене принимаются любые направления, но желательно использовать компьютерный звук.

Demo — это больше, чем просто звук и видео — это гармония, это отражение души, настроения и состояния авторов. Одна только музыка является отражением души человека, и по пристрастиям человека можно определить даже характер, который может быть, в принципе, изменчивым. Например, в детстве я любил песенки Шаинского, затем техно, рэп, Happy Hardcore, рок, а в последнее время опустился до попсы. Видимо жизнь стало попсой, а иногда и попой (мягко говоря) ☺.

Самое сложное в процессе создания демо-ролика — программирование, потому что для того, чтобы поразить зрителя, необходимо создать нечто неве-

роятно потрясающее и удивительное. Необходимо на компьютере с малой мощностью выдавать такие эффекты, которые только анонсируются производителями видеоускорителей и появятся в аппаратной поддержке лишь через несколько лет. Конечно, для этого необходимо очень хорошо разбираться в математике, особенно в геометрии, тригонометрии, матрицах и некоторых других областях. Мы же постараемся не опускаться до такой "низости", а воспользуемся возможностями современных видеочипов. Даже этого будет достаточно, чтобы реализовать интересные эффекты, и поразить зрителя.

Некоторые задаются вопросом — как разработчики демо-роликов умудряются создать графику лучше, чем в играх? Когда мы поняли, что такое ролик, можно легко получить ответ. Просто игра и ролик — это разные вещи. В роликах не нужен AI (Artificial Intelligence, искусственный интеллект), отображение графики идет по линейному сюжету, что значительно упрощает программирование. Получается, что у Demo меньше производственных расходов, и они могут выполнять больше вычислений в тот момент, когда игры обрабатывают AI-монстров.

Дополнительную информацию о демо-сцене можно найти на сайтах:

- **http://www.scene.org** — наверно самый крупный сервер демо-сцены. На этом сервере находится очень много домашних страничек различных демо-групп, где они выкладывают свои последние работы. А на FTP-сервере **ftp://ftp.scene.org** можно найти работы большинства некогда и ныне знаменитых групп и авторов одиночек. Если хотите посмотреть работы профессиональных групп, то советую заглянуть на **ftp://ftp.scene.org/pub /demos/groups/**;
- **http://www.demoscene.ru** — крупнейший российских портал о демо-сцене.

В качестве "контрольного выстрела" предлагаю запустить программу из папки Demo/kkrieger, размещенной на прилагаемом компакт-диске. Это 3D-игра в стиле Quake, которая занимает со всеми ресурсами менее 100 Кбайт (соответствующий скриншот показан на рис. 1.2). Да, она маленькая и ее можно пройти за день, но графика достаточно приемлемого качества, а размер поражает. Попробуйте создать что-нибудь хоть немного приближенное.

Если вас впечатлила эта игра, то рекомендую заглянуть на сайт разработчиков (**http://www.theprodukkt.com**), где можно найти еще несколько красивых демонстрационных программ.

Во время создания игры .kkrieger разработчики использовали множество методов оптимизации. Я не видел исходного кода и не прогонял ее через отладчик, поэтому могу только догадываться, что они сжали исполняемый файл чем-нибудь вроде PECompact, не использовали растровые текстуры, а генерировали их на лету (создание текстуры кодом отнимет меньше места) и ис-

пользовали в качестве эффектов звук минимального качества. Одни только звуковые эффекты не могут занимать менее 100 Кбайт, если их сделать CD-качества.

Рис. 1.2. Скриншот игры .kkrieger

И все же, для своего размера и при таком качестве получается действительно шедевр, который еще не раз прогремит на всю ИТ-вселенную, если разработчики не бросят этот проект. С другой стороны, если они начнут работать в сторону повышения качества звука и изображений, то размер программы станет сильно увеличиваться и тогда проект потеряет "изюминку". Вот почему игре лучше оставаться такой, как она есть, ведь именно малый ее размер делает работу разработчиков действительно гениальной, и поэтому их можно смело называть хакерами.

1.2. Введение в DirectX

Со времен, когда MS Windows еще не был операционной системой (ОС), а только надстройкой для MS-DOS, в качестве программной основы для работы с графикой использовался интерфейс GDI (Graphic Device Interface, интерфейс графических устройств). На то время это была действительно удачная технология, с помощью которой можно было работать с любой видеокартой. На платформе PC было слишком большое разнообразие видеочипов с различными возможностями, и GDI предоставлял универсальный способ доступа к видеофункциям. Эта технология до сих пор используется в Windows, но в значительно переработанном виде.

Универсальность — это хорошо, но производительность видео оставляла желать лучшего. Когда я впервые увидел игру Doom, то поразился, почему игра может создавать сложнейшие сцены на компьютере с 386-м процессором, а Windows не может? Конечно же, разрешение игры ниже, но и сцены трехмерного мира намного сложнее. Производительность GDI — это черепаха по сравнению с прямым доступом к памяти. Основная проблема GDI кроется в том, что ни одно приложение не может получить прямой доступ к видеокарте и видеобуферу, иначе очень сложно будет реализовать многооконную систему, да и универсальность добавляет ложку дегтя и возможности хорошего видеочипа используются не на все 100%.

Из-за медленной графической системы платформа Windows оказалась непригодной для создания игр. Да, на самом деле, были разработки, ускоряющие графику, но разогнать черепаху даже с помощью установки вентилятора на панцире невозможно.

Чтобы решить проблему скорости, возникла необходимость в создании принципиально новой технологии, которая решала бы три основные проблемы:

1. Приложение должно иметь возможность получить прямой доступ к видеопамяти.
2. Возможности видеокарты нужно использовать на все 100%, тем более что не за горами было появление видеоускорителей, которые сейчас стали нормой, а в настоящее время функциями ускорителя уже наделили и видеокарты.
3. Приложение должно получать максимальные ресурсы процессора. ОС Windows многозадачная система, и процессор разделяется между многими процессорами, а в играх это неприемлемо.

Все эти проблемы достаточно эффективно решаются с помощью DirectX, которому предоставляются максимально возможные ресурсы компьютера и прямой доступ к видеобуферу, если приложение работает в полноэкранном режиме. На мой взгляд, это наиболее простая задача. Если процесс занимает весь экран, то ему можно отдать намного больше ресурсов, чем если приложение работает в окне. В оконном режиме ОС должна прорисовывать все, что находится в фоне, а именно графика является более слабым местом программы.

Наиболее сложная задача — использовать максимальные возможности видеочипа. На PC-совместимые компьютеры сейчас устанавливают чипы NVIDIA GeForce, ATI Radeon, Matrox и др. и все они несовместимы между собой. Одна видеокарта поддерживает одни возможности, а другая — совершенно другие. К тому же, есть еще видеочипы, графические возможности которых минимальны (например, чипы от Intel). Как же добиться универсальности? Можно реализовать в DirectX только те возможности, которые есть во всех видеокартах, но что-то подобное уже было в GDI, и производи-

тельность оказалась минимальной. Можно реализовать то, что посчитаем нужным, но тогда программы будут работать не на всех компьютерах и возникнет множество проблем с видеокартами.

Хороших решений проблемы универсальности всего два:

1. *Предоставить интерфейс ко всем необходимым возможностям*. Если возможность поддерживается видеочипом аппаратно, то использовать ее, если же нет, — реализовывать ее программно.
2. *Реализовать то, что хочется, и для совместимости заставить разработчиков следовать спецификации*.

Изначально фирма Microsoft выбрала первый вариант, потому что видеоускорители стоили дорого и установлены были далеко не на всех компьютерах. Движок DirectX мог работать в двух режимах:

1. *HAL* (Hardware Abstraction Layer, уровень аппаратных абстракций) — этот уровень задействуется, если видеочип поддерживает необходимые функции аппаратно.
2. *HEL* (Hardware Emulation Layer, уровень эмуляции аппаратуры) — этот уровень задействуется, когда необходимая функция не поддерживается аппаратно.

Программный уровень (HEL) в DirectX 9 лучше не использовать в конечных приложениях. Вы можете включать его только на этапе разработки приложения. Простые возможности ускорения графики встроены в большинство современных видеочипов. При эмуляции сложных эффектов будет слишком большая нагрузка на центральный процессор, который может уже не справиться с необходимыми расчетами, и вывод графики станет крайне медленным.

На мой взгляд, DirectX — это одна из лучших технологий для игр. В настоящее время у нее только один серьезный конкурент — OpenGL, но возможности DirectX больше, потому что охватывают не только графику, но и звук, и сеть, а значит, для написания игр у DirectX есть все необходимое.

1.3. Установка и настройка DirectX

Для разработки DirectX-приложений в Delphi необходимо всего лишь иметь в своем распоряжении заголовочные файлы. Вы можете найти все необходимое на прилагаемом компакт-диске в каталоге Common/DirectX. Для Delphi 2005 файлы находятся в подкаталоге D9. С остальными версиями все и так ясно по имени подкаталога.

Содержимое нужного каталога необходимо скопировать в заранее подготовленную папку на вашем компьютере и настроить пути, чтобы Delphi мог их найти.

Внимание!

Не стоит пытаться установить файлы DirectX, как компоненты. Если вы попытаетесь это сделать, то встретите только большое количество ошибок и установка завершится неудачей. Нужно только навести пути, чтобы компилятор Delphi смог найти нужные файлы, и все.

Для наведения путей на заголовочные файлы запустите среду разработки Delphi и выберите меню **Tools | Options** (Инструменты | Опции). Если у вас Delphi 2005, то окно будет выглядеть, как показано на рис. 1.3. Перейдите в раздел **Environment Options | Delphi Options | Library – Win32** (Настройки окружения | Настройки Delphi | Библиотека – Win32). Справа ищем строку **Library path** (Путь для библиотек). В это поле нужно добавить путь к заголовочным файлам. Чтобы проще было добавлять путь, щелкаем по кнопке с точками справа от этой строки. Перед вами появится окно, как показано на рис. 1.4. Внизу окна есть поле ввода, где нужно указать путь к каталогам. Щелкните по кнопке справа от этого поля и в стандартном окне выбора каталогов найдите, где вы расположили заголовочные файлы. Теперь добавьте выбранный путь с помощью кнопки **Add** (Добавить), и можно закрывать все открытые окна, щелкнув соответствующие кнопки **OK**.

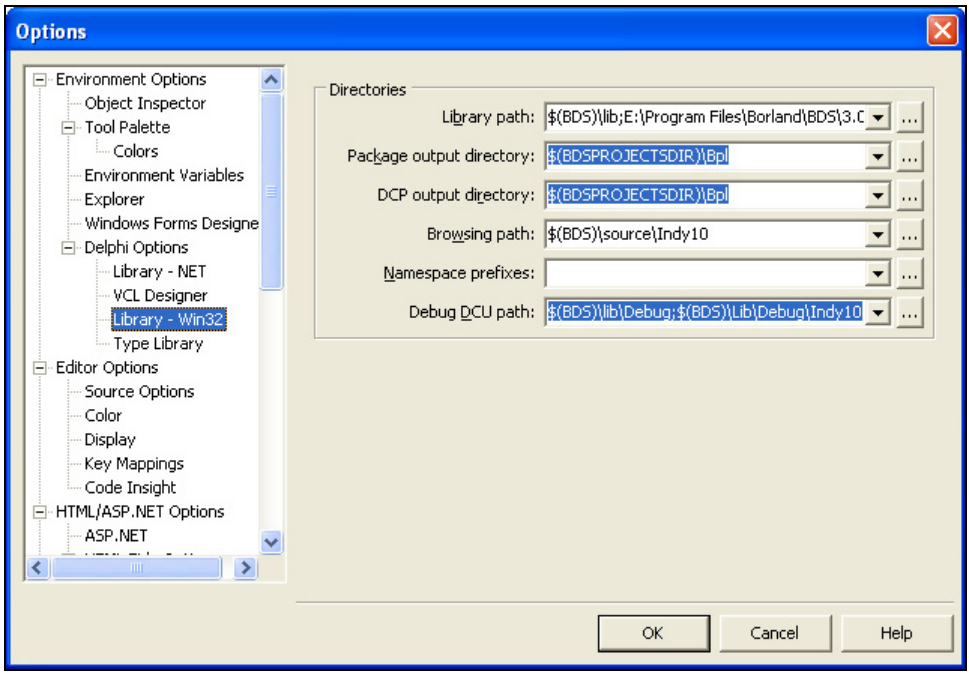

Рис. 1.3. Окно настройки Delphi

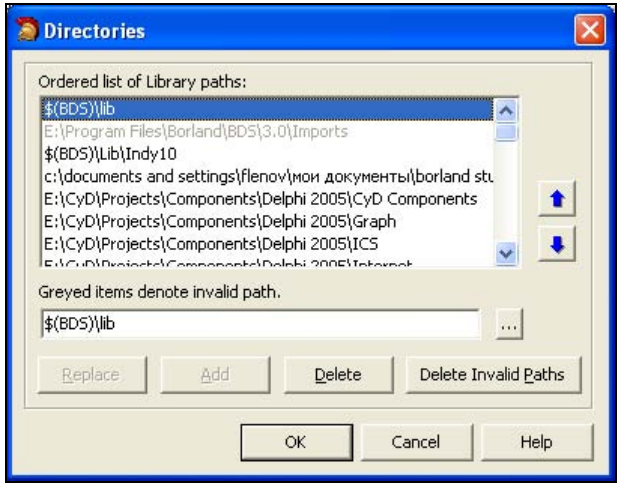

Рис. 1.4. Окно добавления пути

Вот и все. Среда разработки готова к работе над проектами, использующими DirectX.

1.4. Введение в оптимизацию

Я всегда говорю, что если вы написали хорошую программу, то вы программист, а если вы сделали ее лучше других, то вы хакер. Как можно сделать программу лучше других? Один из способов — сделать ее быстрее и не требовательной к ресурсам компьютера. Демо-сцена получила такую популярность именно благодаря оптимизации, которая позволяет выжимать из компьютера все его соки, поэтому мы не можем обойти этот вопрос стороной.

Оптимизация в графике — достаточно сложный процесс, и о различных методах повышения производительности определенных алгоритмов мы будем говорить еще не один раз. Основа любой оптимизации — алгоритм. Если задачу можно решить несколькими способами и выбрать наиболее медленный, то затраты на оптимизацию будут выше, чем эффект от нее. Намного результативнее переписать код на более эффективный алгоритм.

В этом разделе мы опишем основные методы, которым будем следовать при написании кода. Эти методы относятся не только к графике, но и к любым другим приложениям. Об оптимизации можно также почитать в книгах "Программирование на C++ глазами хакера" [1] и "Программирование на Delphi глазами хакера" [2].

Глядя на создаваемые сейчас программы (особенно программистами одиночками), складывается впечатление, что программисты просто забыли про оп-

тимизацию. Программисты наверно думают, что раз их творение в виде исходного кода никто не увидит, то можно писать что угодно. С этой точки зрения программы с открытым исходным кодом имеют большое преимущество, потому что они намного чище и иногда намного быстрее. Создавая код, мы ленимся его оптимизировать не только с точки зрения размера, но и с точки зрения скорости. Глядя на такие вещи, хочется ругаться матом, вот только программа от этого лучше не станет.

Хакеры, однако, тоже далеко не ушли. Если раньше, глядя на программиста или хакера, создавался образ прокуренного, заросшего и немытого молодого человека, то сейчас это цифровое существо, залитое пивом "Балтика" по самые уши, за которого все выполняют машины. Вам медсестра в поликлинике не говорила, что у вас вместо крови одно только пиво льется? Нет, разумеется, я ничего против пива не имею, я и сам его люблю, но надо же и меру знать!

Не надо тратить большие деньги на модернизацию компьютера!!! Начните прежде улучшения с себя. Давайте оптимизируем свою работу и то, что мы делаем, и тогда компьютер заработает намного быстрее.

Итак, рассмотрим законы, которым мы будем следовать при написании кода.

ЗАКОН № 1

Оптимизировать можно все. Даже там, где вам кажется, что все и так работает быстро, можно сделать еще быстрее.

Это действительно так. И этот закон очень сильно проявляется в программировании. Идеального кода не существует. Даже простую операцию сложения 2 + 2 тоже можно оптимизировать (например, использовать сдвиг). Чтобы достичь максимального результата, нужно действовать последовательно и желательно в том порядке, в котором мы будем рассматривать законы.

Помните, что любую задачу можно решить хотя бы двумя способами (или больше), и ваша задача выбрать наилучший метод, который обеспечит желаемую производительность и универсальность.

ЗАКОН № 2

Первое, с чего нужно начинать, — это с поиска самых слабых и медленных мест.

Зачем начинать оптимизацию с того, что и так работает достаточно быстро! Если вы будете оптимизировать сильные места, то можете встретить неожиданные конфликты. Да и эффект будет минимален.

Тут же я вспоминаю пример из своей собственной жизни. Где-то в 1996 году меня посетила одна невероятная идея — написать собственную игру в стиле

Doom. Я не собирался ее делать коммерческой, а хотел только потренировать свои мозги на сообразительность. Четыре месяца невероятного труда, и нечто похожее на движок уже было готово. Я создал один голый уровень, по которому можно было перемещаться, и с чувством гордости побежал по коридорам.

Никаких монстров, дверей и атрибутики на нем не было, а тормоза ощущались достаточно значительные. Тут я представил себе, что будет, если добавить монстров и атрибуты, да еще и наделить все это AI... Вот тут чувство собственного достоинства поникло. Кому нужен движок, который при разрешении 320×200 (тогда это было круто!) в голом виде тормозит со страшной силой? Вот именно...

Понятное дело, что мой виртуальный мир нужно было оптимизировать. Целый месяц я бился над кодом и вылизывал каждый оператор моего движка. Результат — мир стал прорисовываться на 10% быстрей, но тормозить не перестал. И тут я увидел самое слабое место — вывод на экран. Мой движок просчитывал сцены достаточно быстро, а пробоиной был именно вывод изображения. Тогда еще не было шины AGP, и я использовал простую PCI-видеокарту от S3 с 1 Мбайтом памяти.

Пара часов колдовства, и я выжал из PCI все возможное. Откомпилировав движок, я снова загрузился в свой виртуальный мир. Одно нажатие клавиши "вперед", и я очутился у противоположной стены. Никаких тормозов, сумасшедшая скорость просчета и моментальный вывод на экран.

Как видите, моя ошибка была в том, что вначале я неправильно определил слабое место своего движка. Я месяц потратил на оптимизацию математики и что в результате? Мизерные 10% прироста в производительности. Но когда я реально нашел слабое звено, то смог повысить производительность в несколько раз.

Именно поэтому я говорю, что надо начинать оптимизировать только со слабых мест. Если вы ускорите работу самого слабого звена вашей программы, то, может быть, и не понадобится ускорять другие места. Вы можете потратить дни и месяцы на оптимизацию сильных сторон и увеличить производительность только на 10% (что может оказаться недостаточным), или несколько часов на совершенствование слабой части и получить улучшение в 10 раз!

ЗАКОН № 3

Следующим шагом вы должны разобрать все операции по косточкам и выяснить, где происходят регулярно повторяющиеся операции. Начинать оптимизацию нужно именно с них.

Опять начнем рассмотрение этого закона с программирования. Допустим, что у вас есть следующий код (приведена просто логика, а не реальная программа):

1. `А:=А*2;`
2. `Б:=1;`
3. `Х:=Х+Б;`
4. `Б:=Б+1;`
5. `Если Б<100, то перейти на шаг 3.`

Любой программист скажет, что здесь слабым местом является первая строка, потому что там используется умножение. Это действительно так. Умножение всегда выполняется дольше, и если заменить его на сложение (`А:=А+А`) или еще лучше на сдвиг, то вы выиграете пару тактов процессорного времени. Но это только пара тактов и для процессора это будет незаметно.

Теперь посмотрите еще раз на наш код. Больше ничего не видите? А я вижу. В этом коде используется цикл: "Пока Б<100, то будет выполняться операция Х:=Х+Б". Это значит, что процессору придется выполнить 100 переходов с шага 5 на шаг 3. А это уже не мало. Как можно здесь что-то оптимизировать? Очень легко. Здесь у нас внутри цикла выполняется две строки: 3 и 4. А что если мы внутри цикла размножим их 2 раза:

2. `Б:=1;`
3. `Х:=Х+Б;`
4. `Б:=Б+1;`
5. `Х:=Х+Б;`
6. `Б:=Б+1;`
7. `Если Б<50, то перейти на шаг 3.`

Здесь мы разложили цикл на более маленький. Вторую и третью операцию мы повторили два раза. Это значит, что за один проход цикла выполняются два раза строки 3 и 4, и только после этого произойдет переход на строку 3, чтобы повторить операцию. Такой цикл уже нужно повторить только 50 раз (потому что за один раз выполняется два действия). Это значит, что мы сэкономили 50 операций переходов. Неплохо? А это уже несколько сотен тактов процессорного времени.

А что если внутри цикла написать строки 2 и 3 десять раз. Это значит, что за один проход цикла строки 2 и 3 будут вычисляться 10 раз, и мне понадобится повторить такой цикл только 10 раз, чтобы получить в результате 100. А это уже экономия 90 операций переходов.

Недостаток этого подхода — увеличился код нашей программы, зато повысилась скорость, и очень значительно. Этот подход очень хорош, но им не

стоит злоупотреблять. С одной стороны, увеличивается скорость, а с другой — размер. А большой размер — это враг любой программы. Поэтому надо находить золотую середину.

ЗАКОН № 4

(Этот закон — расширение предыдущего)

Оптимизировать одноразовые операции — это только потеря времени. Сто раз подумай, прежде чем начать мучиться с редкими операциями людей, которые ленятся что-нибудь делать.

Так вот именно здесь вы можете проявлять свою врожденную леность в полном объеме. В данном случае крутым считается не тот, кто целый день промучился и ничего не добился, а тот, кто выполнил свою работу наиболее быстро и эффективно. И эти две вещи путать нельзя.

ЗАКОН № 5

Нужно знать "внутренности" компьютера и принципы его работы. Чем лучше вы знаете, каким образом компьютер будет выполнять ваш код, тем лучше вы сможете его оптимизировать.

Этот закон относится только к программированию. Тут трудно привести полный набор готовых решений, но некоторые приемы я постараюсь описать.

- Старайтесь поменьше использовать вычисления с плавающей запятой. Любые операции с целыми числами выполняются в несколько раз быстрее.
- Операции умножения и тем более деления также выполняются достаточно долго. Если вам нужно умножить какое-то число на 3, то для процессора будет легче три раза сложить одно и то же число, чем выполнить умножение. Хотя современные процессоры работают с умножением уже достаточно быстро и разница уже не так заметна.

  А как же тогда экономить на делении? Вот тут нужно знать математику. У процессора есть такая операция, как сдвиг. Вы должны знать, что процессор "думает" в двоичной системе, и числа в компьютере хранятся именно в ней. Например, число 198 для процессора будет выглядеть как 11000110. Теперь посмотрим, как работают операции сдвига.

  Сдвиг вправо — если сдвинуть число 11000110 вправо на одну позицию, то последняя цифра исчезнет и останется только 1100011. Теперь введите это число в калькулятор и переведите его в десятичную систему. Ваш результат должен быть 99. Как видите — это ровно половина числа 198. Вы-