Проект

-

Документация, размышления на тему, идеи алгоритмов.

© [redacted], aka Cheb.

Москва,

1995 - 200?.

Содержание

Введение

Цель этого проекта - попытаться создать (а если не получится - хоть наметить пути для других первопроходцев) игру, совмещающую в себе свойства несерьёзной аркады и сложной RPG, помноженные на увеличенный реализм.

Ну, и одновременно, основать принципиально новый трёхмерный "движок", который был бы лишён недостатков, свойственных большинству современных, чьё развитие, по моему мнению, зашло если не в тупик, то в очень узкое место.

При внимательном рассмотрении современной ситуации с играми разных жанров, обнаруживается некоторая пустота. Казалось бы, уже созданы игрушки всех жанров - и шуттеры, и стратегии, и RPG, и - в последнее время - самые замысловатые их гибриды... Одного не видно - настоящей виртуальной реальности. Такой, чтобы заставляла почувствовать самого себя "там" - не мультперсонажем, прыгающим как мячик с одним процентом здоровья, и не супергероем с кучей характеристик, превращающих игру в математическую олимпиаду, а реальным человеком из плоти и крови.

Шуттеры закостенели в своей неподвижности, совершенствуя лишь графику. Их физические законы не изменились ни на грамм с времён Doom'a, но лишь оттачивались в тонкостях, приближаясь к своему, очень далёкому от реальной жизни идеалу, где сошедшиеся в дезматче геймеры убивают друг друга каждые несколько минут, чтобы тут же воскреснуть. В сущности, этот игровой мир абсолютно условен - математически совершенное воплощение правил детской игры в войну.

В RPG иллюзия реальности достигается несколько иным способом - тщательно скроенный, заботливо прокачанный, и проведённый через множество трудных мест герой становится таким привычным и родным, что спустя часы, дни и недели, проведённые за дисплеем, невольно начинаешь воспринимать его, чуть ли не как себя самого - подсознательно принимая за реальность причудливую арифметику RPGшного мира. Хотя, если посмотреть на тот же знаменитый Fallout... сколько, думаете, пуль надо всадить человеку в глаз, чтобы он умер? Обычному и двух хватит, а вот китайцу... Короче, реализма мало.

Реальной виртуальность делает наш собственный мозг, который, как оказалось, с лёгкостью приспосабливается к самым невероятным условиям, привыкая к ним, включая в круг привычных вещей. Но есть тут одно "но". Чем больше физика игрового мира оличается от реальной, тем больше наше заэкранное воплощение отличается от нас самих. Тот, кто бежит по коридорам и лабиринтам Квейка, отстреливаясь, и шарахаясь за угол - да, это действительно я сам, и моё покинутое тело за дисплеем невольно повторяет те же дивижения... Но это совсем не тот я сам, кто отправляется каждое утро в институт, прогуливать лекции, или отрабатывать повтор. Налицо две разных личности, с разными системами ценностей. Очередью прошили?.. Ерунда, аптечка всё поправит за долю секунды! Главное - успеть добежать, и наступить на неё... По лаве пробежаться?.. Ну, циферка уменьшилась. Ну, не до нуля же...

Приспосабливаясь к ненормальным условиям, мозг перестраивается сам, меняется система ценностей и ощущений. И бочка мёда - погружение в виртуальность - оказывается приправленной ложкой дёгтя - излишней нереальностью нашего заэкранного "я", граничащей с раздвоением личности.

Почему же, почему подавляющее большинство игр грешит такой явной ненатуральностью?.. Мне кажется, дело в том, что создатели игр, как бы они ни любили своё детище, скованы жёсткими законами рынка. Тут видна прямая аналогия с голливудскими фильмами: пусть герой из пистолета валит толпу автоматчиков, внезапно охваченных острым приступом кривоглазия и косорукости, пусть сюжет и диалоги тихо скатываются к полному маразму... Главное - зрелищность! Взорвать поярче, в морду дать поэффектнее, природный ландшафт снять покрасивее. Иначе ведь купят не твоё, а конкурентово - и будет тебе волчий билет на всю оставшуюся жизнь. И редко-редко появляются в массе сией стоящие вещи.

А с играми ситуация ещё хуже - в плане потребителя, на ублажение которого всё и ориентировано - а судьи кто?.. Правильно, дети в массе своей. Размером от среднего до крупного. И тот, кто им не угождает - теряет позиции, и сметается с рынка. Грустно...

Но нет-нет, да и сверкнёт луч света в тёмном царстве. Из сонма играных мною игр приведу два примера, разными путями пытающихся приблизиться к настоящей реальности, и в известной мере, полярных. Это две крайности, лежащие по разные стороны игрового сонмища:

1.) Нет никакого "воплощения". Идёт погружение самого играющего в виртуальный мир, за счёт тщательно созданного эффекта присутствия. Единственный известный мне пример - дилогия Myst - Riven от Cyan:

С первого взгляда, Ривен (5CD) - это набор слегка оживлённых слайдов, с вкраплёнными кое-где героями, заранее снятыми на видео. Но эта игра - в гораздо большей мере виртуальная реальность, чем многие трёхмерные бегалки-прыгалки, где в реальном времени графика ненамного хуже.

И дело здесь вот в чём: фантастическй звук, великолепные ландшафты, прорисованные на мощной графической станции - воздействуя на органы чувств, создают эффект присутствия, с одной стороны. А вот физика этого мира существует исключительно в воображении игрока. Я щелкаю мышью по экрану, и одна фотография плавно сменяется другой - и уже моё воображение рисует мне, как я иду по острову, ведь никакого объекта, моделирующего моё виртуальное тело, в игре не существует...

Я заэкранный на Ривене - тот же, что я в жизни. Я не могу лазать по отвесным скалам, хотя и хотелось бы. Я не желаю замочить ног, я ни за что не полезу туда, где можно пораниться, я ни с кем не дерусь, и пытаюсь одолеть злодея хитростью. Если в меня стреляют, я умираю на месте...

Тут уж, как ни крутись, любая ВР будет иметь свойство обратимости - даже я, с моей тягой к реализму, не стал бы играть в игру, где нет Save/Load.

Чем ещё подкупает Мист - так это скромностью поставленной задачи. Не спасаешь целый мир, не борешься со вселенским злом, а просто помогаешь хорошему человеку, и попутно ищешь путь домой. В Ривене, правда, видна деградация - не просто ищешь выход, но и спасаешь около тысячи островитян от гибели. Хотя спасать

их и не обязательно... Пять вариантов конца игры (из которых три смертельных) - это не шутка!

2.) Воплощение игрока в виртуальном мире настолько глубоко проработано, что начинает казаться собственным телом. Единственный известный мне пример - Robinson's Requiem от Silmarils.

Сделать скриншот с Робинзона - задача не для смертного.
Под Windows он сразу "GP Fault" - и лапки кверху,
а под голым ДОСом защита могуча и непробиваема.

На фоне Ривена, Реквием по Робинзону выглядит приветом из глубины веков (шло на 386 с 4Мб памяти). Графика для тех времён - революционная (огромное открытое пространство со сложнейшим рельефом, трава- действительно пучками торчит вверх, и найти в ней копьё после промаха по бизону весьма непросто - не то, что на камнях). Жаль только, реализовано это было жутко криво, дискретности так и пёрли из всех щелей... Но да чёрт с ней, с графикой! Графика в хорошей игре - совершенно не главное.

Начать, хотя-бы с сюжета - совершенно нетрадиционного. Квесты на выживание, оформленные, как 3d-action, мне с тех пор больше не попадались... И глвная цель игры - спасти только себя, любимого, то есть выжить, и убраться с этой треклятой планеты. Есть, правда, ещё телепатка Нина - обладательница единственного неразбитого звездолёта, но она проявляется только во сне, и в финальной заставке.

Главный же козырь Реквиема - крайняя реалистичность самого игрока. Это сложная модель человеческого организма, в которой учитываются такие мелочи, как количество воды, крови, питательных веществ, наличие ран и болезней, температура тела и частота пульса (простудился - ешь аспирин неделю, пока не перестанешь кашлять), и необходимость подбирать одежду к погодным условиям... То есть, даже пройти через два ландшафта к третьему непросто, надо всё время заботиться о еде и ночлеге, обороне от хищников...

Конечно, о полном реализме речь не идёт. время там сжато - сутки проходят примерно за полчаса, раны под повязками заживают подозрительно быстро даже на высшем уровне сложности... Но если рану не перебинтовать - можно умереть от потери крови. Если рана глубокая - лучше зашить её, бинты не сразу помогут. Но если шить по живому, без наркоза (звуковое сопровождение - выразительный вопль), то на второй ране умрёшь от болевого шока, а новокаина всего десять ампул на всю игру. Если началась гангрена - остаётся загрузить игру по новой, или ампутировать конечность. Если бизон приложил слишком сильно, и рука (нога) сломана - игру не пройти. Можно, конечно, наложить шины, и ждать пока срастётся... Лично у меня никогда терпения не хватало - ждать, пока на экране пройдут месяцы. (Ведь всё это времяя надо чем-то питаться, чтобы не умереть от голода, а это занятие - очень нудное.)

Короче, дня не хватит, чтобы рассказать обо всех тонкостях и нюансах, заложенных в эту невероятную игру, равных которой с тех пор не создано. (А может, просто не скомуниждено нашими ленивыми пиратами).

Итак, исходя из всего вышеизложенного, какой путь мне избрать, работая над этим проектом ?..

Из двух описанных выше, первый практически исчерпан на современном этапе развития техники, и требует поистине титанических затрат труда. Команда, которая осмелится пойти по нему, получит лишь более или менее удачное повторение Ривена (на 5-10 DVD дисках, полагаю).

Второй путь - единственный пригодный для скрещивания с трёхмерной аркадой, и является, фактически, небольшой навеской на её физическое ядро и интерфейс. Именно это направление я выбираю для себя, поскольку других мне не дано.

К истории вопроса

(моя автобиография, как программиста)

Сначала был Бэйсик и мкро-ЭВМ "Искра-226", и я нёсся пустынными коридорами маминого НИИ, нетерпеливо размахивая дискетой размером с блюдо, ибо до машинного зала было далеко:

```
5 DIM J%(100,10)10
10 PRINT HEX(0307):%OPEN J%():INPUT "НАЧ.СКОРОСТЬ",N:INPUT "УГОЛ",U:U=
15 SELECT D
20 V1=N*COS(U):V2=N*SIN(U):W=2*V2/9.8:D=V1*W:D=-D
25 DRAW J%(),511,0:DOT J%(),0,0
30 FOR T=1TOD-3STEP0.01
40 X=ABS(T*V2):Y=ABS(V1*T-9.8*T^2/2):IF X>DTHEN70
50 DRAW J%(),X,Y
60 NEXT T
70 PRINT "DAL_NOST_ POLETA=";-D:END
```

Потом там поставили более современные, хотя и редкие в те времена, IBM-совместимые компьютеры с цветным дисплеем, на которых я баловался с графикой, и написал для этого графический редактор. (естественно, управляемый

только с клавиатуры) Постепенно я понял, что QuickBasic'y доступны только 40 килобайт памяти из 640 имеющихся, и стал искать более современный язык. И попался мне Borland Turbo Pascal 5.0, хотя лучше бы я тогда наткнулся на C... Но сделанного не воротишь - изучить другой язык, кроме Паскаля, у меня с тех пор руки так и не дошли. (а Бэйсик я совсем забыл за ненадобностью)

К попыткам написать собственную игру я приступил довольно давно - ещё в лохматом 95-м, когда у меня появился собственный мощный компьютер, с 386 процессором и 4Мб памяти. Windows тогда считалась - да собственно, и была - большой-пребольшой бякой, и называлась маздаем, а про Windows 95 если кто и слышал, так только страшные истории о 95 дискетах..

Попытки создания собственной игры были многочисленны, как тараканы, но лишь однажды увенчались скромным успехом, породив недоделаный, но забавный шуттер с одним уровнем, и одним монстром на нём. Да, Змей Горыныч мне тогда удался неплохо... Потом туда, правда, добавилась парочка монстров, спёртых из doom'a, но преображённых до неузнаваемости, и различные варианты травы, кустов и деревьев, красиво разлетавшихся в щепки при попадании из гранатомёта...

А потом эпоха спрайтованной графики безвозвратно канула в Лету, а игра отправилась в архив вместе с так и не доделанным алгоритмом искуственного интелекта монстров.

Попытки эти проваливались, в основном, из-за быстро растущей требовательности к своим творениям, заставлявшей проходить путь раз за разом сначала. Внезапно полученная порция свежей информации (а Интернета тогда у меня не было, и нужные сведения приходилось добывать потом и кровью) каждый раз приводила к одному результату: я убеждался, какая, в сущности, устаревшая на корню гадость у меня прорисовывается, и делал почти то же самое заново.

Первая уродина родилась, когда я чудом узнал, как использовать EMS. (на лабораторной, которую делали на Искре-1030 (стояли тогда ещё эти динозавры в родном МИИТе), пошарил на винте, и нашёл давно искомую документацию).

Это была (бы) полноценная система управления виртуальной памятью, позволявшая динамически выделять и освобождать блоки, и изменять подключенную базу данных в процессе работы. Плюс вживленная в неё поддержка многозадачного режима с блокировкой критических секций (одна процедура выполнялась в цикле, а вторая - параллельно, в остатке времени цикла, ничего не зная о переключениях). Плюс маниакальная виртуальная видеосистема, с идентичной структурой фреймбуфера для совершенно разных VGA, X-VGA, и SVGA видеорежимов, и 4096 градациями яркости со встроенным дизерингом.

После года проведённого за отладкой и совершенствованием этого чудовища, я затосковал без реально видимых результатов, упростил всё по чёрному, и построил бассейн на фундаменте Дворца Советов. Так родилась уродина вторая (долго совешенствовавшаяся, но в конце концов, тоже вымершая). Там не было настоящей виртуальной памяти, только внешняя библиотека с фиксированным размером страницы 16К, равным странице EMS. Спрайты были только 128x128, звуки - не длиннее 1,45 секунды (это позднее, когда я купил библиотечку для работы с Sound Blaster'ом). Видеорежимов она поддерживала аж целых два: 320x200x8 и 320x200x16, с табличным фоггингом. Ну, и изюминка - стены, набранные из вертикально вытянутых спрайтов. Там вообще не было ничего, кроме спрайтов, заботливо сортируемых по глубине... Отдельная песня - как я пытался присобачить к ней покупную библиотеку к C++ для работы с саундблястером. Тогда же, я был вынужден написать свою первую программу на C, и окончательно возненавидел этот мутный язык. В результате, родился жуткий гибрид: писаный на C резидентный драйвер блястера, общающийся с основной паскалевской программой через прерывание и специальный буфер.

А потом случилось страшное - я повёз свой шедевр показывать Лёше, и на его сверхмощном 486DX66 эта дрянь *тормозила*!..

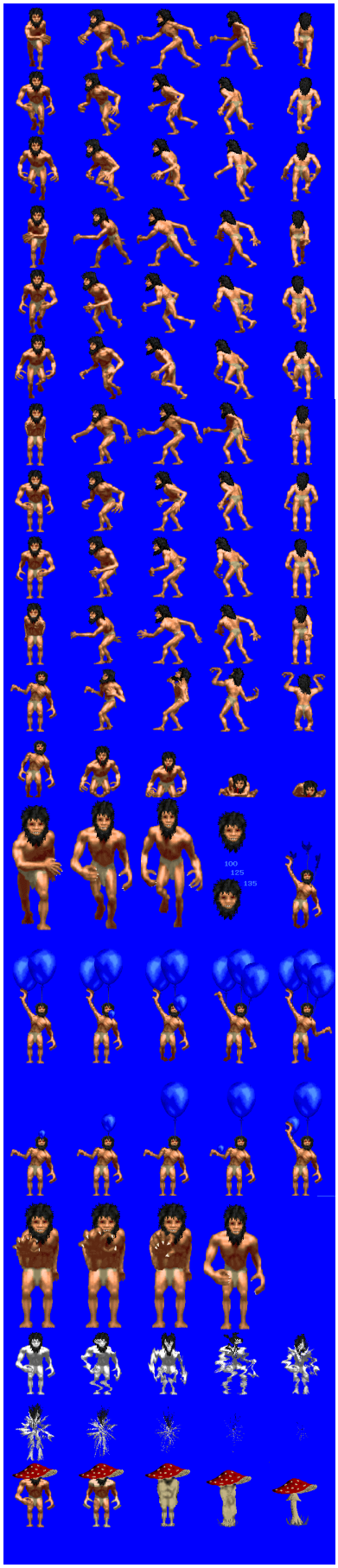
Нет, в чистом поле, среди ёлок, всё было отлично. Но стоило зайти в замок (всего-то полтысячи спрайтов вокруг), как частота кадров резко падала. Я ещё мог понять это на моём 386-м, который давно не был венцом прогресса, но теперь... И я забросил весь проект на год или два, занимаясь мелкой чепухой.

Потом я получил хороший урок, попытавшись использовать объектно - ориентированный код, и убедившись, *как* он тормозит. (без видимых причин, между прочим) Урок я усвоил настолько хорошо, что даже теперь, с удовольствием работая на Дельфи, путаюсь во всех этих классах и объектах.

Потом, спустя время, я вернулся к старому проекту. Трёшки уже были безнадёжным старьём, я резвился на четвёрке, и философски рассуждал, что пока я закончу игру - глядишь, все на Пентиумы пересядем, и останется торможение только в воспоминаниях.

Потом начались битвы за скорость, и совместимость, сопровождавшиеся постепенным наращиванием графики (в количественном смысле). Сначала выяснилось, что встроенный диспетчер EMS в Windows тормозит по чёрному без объяснимых причин. Пришлось приделывать модуль распознания "форточек", который ругался, и требовал запустить игру в голом ДОСе. Потом начались загадочные зависания. (как известно, самая мерзкая разновидность неисправности - перемежающаяся) После очень долгой и тяжёлой отладки (ведь паскалевский отладчик моей многозадачной программой давился насмерть, так что приходилось использовать подручные средства) выяснилось, что висяки случаются тогда, когда звук кончается, или сменяется другим. Ох, скольких нервов это мне стоило, и скольких часов за отладкой, и грызением локтей... Я уже и многозадачный модуль вырезал... А потом, действительно по-подлому, выяснилось что моя чудесная 486-я материнская плата конфликтует со всеми звуковыми картами. (видать, кривой уродилась) И программа-то ни при чём.

Следующий год я боролся со звуком, сооружая многоканальный миксер. Сначала использовал имевшуюся библиотеку, запуская звук раз в секунду. Результат получился неплох, несмотря на регулярные щелчки. Позже, когда Лёша подключился к Интернету, и нашёл мне инструкцию по бластеру, я переделал звуковой модуль, использовав auto-init DMA. Но ничто уже не помогало скрасить убогость общих принципов: как бы ни была прикольна сооружаемая мной аркада, каждого монстра следовало прорисовать с пяти точек зрения, в десяти разных фазах движения - задача, может, и простая для художника - профессионала, но чересчур тяжёлая и нудная для меня.



Жанр выдыхался. Наворотов становилось всё больше, а монстров не прибавлялось. Поголовно 16-килобайтные, спрайты выжирали память вчистую, и даже модернизация базы данных с поддержкой более длинных звуков уже не радовала...

И наступил момент, когда я понял, что тема исчерпана. И родилась уродина третья. Шёл 98 год, Windows 95 OSR2 победно шествовала по миру, захватывая страны и континенты. Старая игра задыхалась от нехватки базовой памяти, которой под Маздаем не то что 640, а и 400 Кбайт не всегда имелось...

И вот, в один чудесный день, безнадёжно пуская слюни над Борланд паскалем для защищённого режима, я случайно узрел в одной из программ - примеров волшебное слово, распахнувшее мне двери в рай: Seg0040. Дрожащими руками я набрал тестовую программу.. Я не ошибся: SegA000 тоже существовал! И действовал!.. Теперь я мог дотянуться до видеопамяти из защищённого режима. И закипела работа...

Программа, порождённая Борланд-паскалем, была абсолютно живуча. Она с равным успехом шла как из Windows, так и из "чистой" конфигурации, она не зависела ни от каких QEMM'ов и прочей мутоты, ей нужны были лишь 160 Кбайт базовой памяти для взлёта, вся остальная память была доступна для динамических переменных. Она, с моей небольшой доработкой обработчика ошибок, была полностью защищена от исключения 13, то есть от внутренних ляпов. Это была идеальная, совершенно демократичная среда, годившаяся как для хилой "четвёрки" с ДОС 6, так и для навороченного "Пентиума" с Windows 98... Возникавшие неразрешимые проблемы я, окрылённый, щёлкал, как семечки: Не дотянуться до физической памяти, чтобы настроить DMA саундбластера ?.. Вот вам драйвер

реального режима ДОС, запускаемый программой, общающийся с ней через прерывание, получающий данные через видеопамять, которая на всех одна. А вот отлично вписался в новую систему извлечённый из пыльного сундука модуль многозадачного режима... В чистой конфигурации нет мыши?.. Вот вам модуль, ищущий Mouse.com по всем дискам, а при случае, и свой из кармана достанем! Ограничение в 64Кбайт?.. Ничего, потерпим!.. Сборщик базы данных работал под Windows, используя все её 32-битные преимущества. Ну, а для самой программы есть Турбо-ассемблер, в котором 32-битный код писать - одно удовольствие. Могучий фундамент всё рос и рос, день ото дня становясь всё шире и надёжнее. Двуязыковая поддержка, подробные сообщения обо всех мыслимых и немыслимых ошибках, модуль работы с джойстиком с автокалибровкой, автоопределением порта и защитой от зависания при отрыве разъема... Да, это была вещь на века, достойная войти в двадцать первый век, несмотря на лёгкую архаичность фундамента!

Конец был быстрым, и печальным. Однажды вечером, когда ничто не предвещало угрозы, я сидел и читал на сон грядущий скачанную из Интернета Интеловское руководство по оптимизации кода. И случилось страшное...

Давно известно, что защищённый режим Борланд-паскаля - с 16битной адресацией, и 16-битными операндами. И чтобы использовать в нём неродную 32битную адресацию, надо поставить перед командой префикс 67h. То же и для работы с 32битными операндами - префикс 66h. На 386 или 486 процессоре это работает превосходно. Но вот мудрый Пентиум размышляет над каждым префиксом три такта. Три плюс три равно шесть - вместо того, чтобы выполнять две команды за такт.

И понял я, что мой великолепный фундамент обречён тормозить. И предал его забвению, забросив раз и навсегда.

И началось долгое, кропотливое освоение Delphi и Windows95, которая мерзка тем, что настоящий многозадачный режим (как в моих первых попытках) под ней не построишь, ибо крива она, и уродлива. Но что самое печальное - все годами копленные знания матчасти и умения их использования оказались здесь совершенно бесполезными.

***

25.02.2000. Что-то сегодня нет настроения отлаживать свежесляпанный перехватчик клавиатуры & мыши... Зато было настроение выколупать шрифт (3..5)x7

из старой игры (исходник, конечно, давно потерян), и нарисовать его удвоенный (6..10)x14 вариант - пригодится. Я ведь по-прежнему не знаю, как в Windows узнать ширину выводимой строки текста в пикселях.

26.02.2000. Ура, наконец-то удалось дотянуться до видеопамяти! Ларчик, конечно, просто открывался: TDXDraw.Surface.Lock, как один из параметров, возвращает указатель на отображённую видеопамять - и никакого контроля! Указав по ошибке размер буфера в 8 раз больше, я случайно раскрасил половину десктопа процедурой FillChar - и это при том, что программа работала в окне, а не на весь экран.

Спасибо, конечно, дорогому товарищу Хироюки за DelphiX, но без английского хэлпа приходится слишком долго мудохаться, экспериментальным путём изобретая велосипед...

04.03.2000. Долго я колупался, пытаясь создать объект типа TDXDraw внутри стартовой процедуры, но ничего не вышло. Пришлось вставлять его прямо в форму, визуальным методом... Грызёт меня сомненье: если запустить игру на машине без DirectX, не вылетит ли она ещё при запуске ?

14.03.2000. Наконец-то выяснил причину нехороших зависаний перехватчика. Оказывается, система "забывает" сообщить об отпускании клавиши Alt. В результате, нажатие на следующую клавишу оказывается "управляющим", со всеми вытекающими. Кажется, Alt вообще действует, как кнопка с фиксацией - придётся запрещать её, или искать какой-то обход.

21.03.2000. А вот когда перехватчик не передаёт событие дальше по цепочке, Alt работает нормально...

12.05.2000. Сегодня нарисовал первую половину анимированной эмблемы CGE (появление).

Составляющие, необходимые для реализации проекта

1). Движок, серьёзно оптимизированный под RPG / квест. Конкретно, поддержка бесконечной местности и очень широкого набора моделей, чтобы персонажи не повторялись.

Например, для каждой модели - 16 голов х 16 текстур х 4 цветовые карты = 1024 разновидности !

2). Узко специализированный трёхмерный редактор, работающий на прнципах объектно - ориентированного программирования: наследование движений от модели - предка, вариантность сменяемых частей (типа голов, рук, причёсок, доспехов и пр.).

3). Набор специализированных утилит, для быстрой и удобной сборки мира и сюжетов

1. Game engine: "мы пойдём своим путём"

1.1. Обзор положения.

Истинно сказано: "Если в Doom'e монстры при ближайшем рассмотрении оказывались набором квадратиков, то в Quak'e - набором многоугольников"...

И никакая умелая раскраска не могла скрыть квадратности их мерзких физиономий. Проковырявшись несколько лет на игровом поприще, я отлично понимаю, что это - печальные следствия хилости компьютерного "железа", неспособного на большее, сначала от нехватки памяти, потом - от нехватки быстродействия. А недальновидное человечество, спешно наращивая могучесть графики, и обходя аппаратные ограничения, зашло в нехороший тупик:

Игра, не поддерживающая 3d-ускорители, обречена на вымирание, так как уступает конкурентам в красивости и частоте кадров. Игра же, поддерживающая ускорители, выживает, но втискиваясь в прокрустово ложе жёстких ограничений, калечится и кривеет: несмотря на свои огромные возможности в плане вычисления световых эффектов, ускорители имеют больше недостатков, чем преимуществ: все объекты должны быть представлены наборами плоских многоугольников, и чем больше их в кадре, тем сильнее акселератор тормозит, и полезность его стремится к нулю. И вот несчастный игродел, в погоне за скоростью, старается собрать уровень или персонажа из как можно меньшего числа плоскостей, искусно их раскрашивая для маскировки. В результате получается бред: некоторые объекты, вроде зданий, получаются как живые, а вот мусорное ведро на их фоне нагло выпирает своей шестигранностью... Люди, конечно, в этой системе выходят не лучшим образом, усовершенствованная реалистичность освещённости только выпячивает корявость моделей... Не говоря уже о играх, где действие развивается на свежем воздухе, и дальние холмы, при удалении от них, внезапно исчезают, открывая взору плывущие по небу облака. В лучшем случае, если у создателя игры была совесть, дали скрыты густейшим туманом - но тогда, опять же, неба не видно.

1.2. Предложения по существу.

Так какой же должна быть действительно хорошая графика ?..

Во-первых, изощрённая освещённость желательна, но совсем не обязательна. Меньше всего человек замечает неправильностей именно в освещении, (пример: не кажется удивительным, что в пещере не абсолютно темно, и всё видно) так что, это - первое, чем можно пожертвовать.

Во-вторых, все объекты должны быть очень высоко детализованы, и не распадаться на плоскую геометрию, даже если ткнёшься в них носом - но и не занимать лишних ресурсов системы, экономно теряя детали по мере удаления.

В-третьих, при наличии неба видимость должна быть безграничной. (т.е. ограниченной только размерами игрового мира, или линией горизонта) - на первый взгляд, реализовать невозможно, но если синтезировать ландшафт воксельным методом, включая в него крупные объекты типа деревьев или зданий, и отображая мелкие одним пикселом, то задача вполне поддаётся решению.

В-четвёртых, при наличии солнца все объекты должны отбрасывать достаточно достоверные тени - это, как мне кажется, реализовать на современном этапе практически невозможно, нужен Ray tracing.

Итак, накопившиеся у меня идеи таковы:

1. К чёртовой матери акселераторы! Если ограничить размер экрана 640x480, как в Dungeon Keeper II, то мощного и навороченного современного компьютера хватит на что угодно, без всякого ускорителя. А не мощный и не навороченный потянет то же самое при 320x240. К тому же, можно вспомнить хитрости наших дедушек, писавших игры для 80286, или безвестного бармена, разоблачённого Джеймсом Бондом ("берётся такой вот стакан с толстенным дном, и огромная маслина...")

Короче, 3d - окно занимает серединку экрана (62% по размеру, или 40% от площади), а вокруг располагаются густым роем различные кнопки, пиктограмки, статусбары, и вокруг всего этого - толстые и красивые рамочки... И даже кажется, что так и надо.

2. Модель должна состоять из криволинейных поверхностей, при повышении детализации апроксимируемых всё большим числом треугольников. Причём, чтобы её можно было отобразить и без апроксимации, плоскими гранями. Это сделать проще, чем некоторые думают: строится модель из плоских треугольников, для каждой вершины задаётся вектор нормали (усреднённый, по соседним треугольникам, или заданный вручную - неважно). А затем предполагается, что стороны треугольников криволинейны, и у каждой вершины составляют с нормалью прямой угол. Если средняя точка кривой слишком отстоит от средней точки отрезка, её делят напополам. (и одновременно - треугольник) И у попа была собака, рекурсивным методом.

3. Нарисовать тьмы и тьмы высокодетализированных объектов для хилой машины будет затруднительно, из-за большого объёма вычислений. Тут на помощь придёт воксельное описание, или иными словами - трёхмерные спрайты, которые на мой взгляд, применялись в истории незаслуженно мало. Конечно, формировать их из 3d-моделей надо ещё на этапе компиляции базы данных игры, и применять к тому же mipmapping.

Воксельный объект: дёшево, и сердито
(Blood-1, на моторе Дюк Нюкема)

А вблизи - всё те же квадратики.
Мерзкое зрелище...

4. Даже применение воксельных объектов не спасает от весьма неприятного ограничения:

Имеем густой лес, в худшем случае - то есть, смотрим на него с холма, и ни одно дерево не закрыто складками местности. Плоская равнина, покрытая лесом, уходит за горизонт. Где-то там, в дали, где их видимый размер меньше одного пиксела, деревья сливаются в одно зелёное покрывало, существующее в виде двумерной таблицы добавлений к рельефу местности (цвет считать зелёным, Δh такое-то), и вся эта радость,

теряя вчетверо при каждом удалении вдвое, плавно уходит за горизонт (для каждого экранного столбца - около тысячи воксельных операций в потоке, при дальности горизонта в 30 километров). А вот что делать с теми деревьями, что расположены ближе ?..

Для начала, вычислим их количество: диаметр дерева - 5 метров, деревья стоят вплотную. Размер экрана 400x300, угол зрения - 112º. Видимый размер дерева равен одному пикселу (0,28º) на расстоянии $\frac{5}{\mathrm{tg}(0{,}28^\circ)} = \frac{5}{0{,}0489} = 1024$м., двум пикселам (0,56º) - на расстоянии 512м, четырём пикселам (1,12º) - на расстоянии 256м, восьми пикселам и более - на расстоянии ≤ 128м.

Приближённо, число деревьев, попадающих в зоны с различной детализацией: $N_{\geq 8}=\left(\frac{128}{5}\right)^2 * \left(\frac{112}{360}\right) \approx 200$. Соответственно, $N_{8\text{-}4} = 600$, $N_{4\text{-}2} = 2500$, и $N_{2\text{-}1} = 9800$. ∑=13000 - ужас, да и только... А ведь при более высоком разрешении, число подлежащих визуализации деревьев может достигать 40 и более тысяч!

Что же с ними, со сволочами, делать?.. С теми, что от 1 до 2 пикселов - вроде, ясно. Включи в воксельное описание рельефа - и порядок...

А если это - двадцатиметровая пальма с шевелюрой на макушке ?.. И нарисуется она, родимая, столбиком 2x8, а при приближении к нему, нижняя часть столбика вдруг исчезнет (при ширине 4 пиксела ствол практически невидим) - налицо мигание, неэстэтичное, и хорошо заметное для глаза. Это сойдёт с рук, разве что, на сотом Пентиуме...

Единственный способ убить обоих зайцев одной пулемётной очередью - применение буферных спрайтов. Воксельные деревья рисуем не прямо во фреймбуфер, а в спрайты, размером вдвое больше нужного - например, 4x16 для зоны (3-1,5пикс.). Предположим, что высота дерева ограничена 20м., и диаметр - 5м., и техника эта не применяется для ближней зоны, где видимая ширина дерева больше 16 пикселей (16x64):

$S_{1,5\text{-}1}=(N_{1,5\text{-}1}*(2x8))=7200*16=115200$,

$S_{3\text{-}1,5}= (N_{3\text{-}1,5}*(4x16))=4350*64=278400$,

$S_{6\text{-}3}= (N_{6\text{-}3}*(8x32))=1090*256=279040$,

$S_{12\text{-}6}= (N_{12\text{-}6}*(16x64))=270*1024=276480$,

$S_{16\text{-}12}= (N_{16\text{-}12}*(32x128))=40*4096=163840$,

Суммарная площадь буферных спрайтов равна:1112960 пикселей, или 927% от площади экрана (120000). Этот буфер зажрёт около 3 Мбайт памяти для круговой панорамы с 8-битными спрайтами.

Казалось бы, имеем десятикратный проигрыш. Но стоит немного прислушаться... Да, это доносятся из ближайших кустов предсмертные вопли третьего зайца - зацепило рикошетом, беднягу...

И этот третий заяц габаритами больше напоминает бегемота.

Буферные спрайты мы не обязаны перерисовывать каждый кадр! Нарисовали - и пусть себе лежат, пока угол зрения значительно не изменится, или солнце не пройдёт по небу большой отрезок. Более того, перерисовкой их можно пожертвовать в критической ситуации. Когда игрок загибается в толпе высокодетализированных монстров - ему не до разглядывания пейзажей. Он жаждет одного - частоты кадров.

Опять же, на хилой машине, вроде того же Пентиума-166, под издевательскую присказку "Апгрейдить надо было вовремя!", можно солидно сэкономить время процессора: перерисовывать деревья, только когда совсем уж припрёт. Ну и что, что они поворачиваются рывками, и на несколько секунд мутнеют при быстром к ним подбегании ?.. А чего Вы, собственно, ждали от вашего дохлого писюка ?!

Но это всё - лирика. А вот физика:

25 тыс. пикселей - для спрайтов рисуемой шириной 2 пиксела (2x8..2x2),

35 тыс. пикселей - для спрайтов большей ширины.

Итого, 50% от площади экрана - в наимерзейшем случае (не заслонённые ничем деревья максимальной высоты, набитые плотно, как кильки в банке).

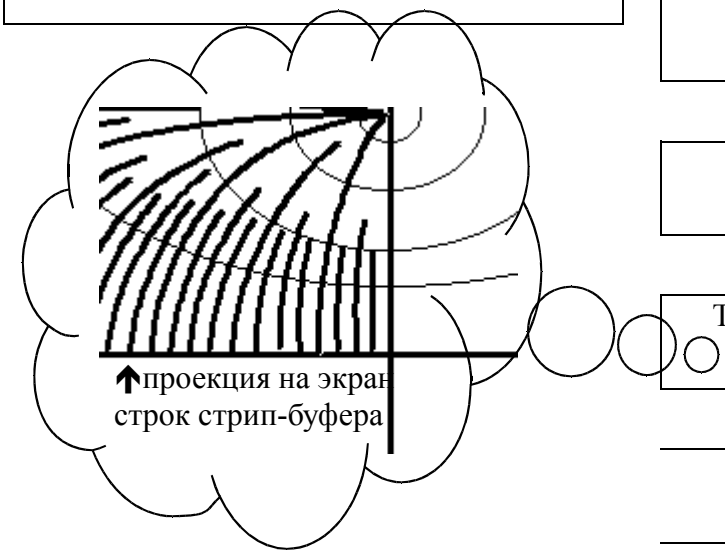
А потом - то же самое, для травы и кустов: полметра ширина, один - высота. Это ещё 25% от площади экрана.

И получится просто фантастическая детализация, под дробный перестук отвисающих челюстей.

5. Был, в своё время, такой чудесный метод - сканирующей плоскости, впервые применённый авторами DOOM'a. Один у него был минус, но фатальный: вертикальные линии всегда параллельны, и посмотреть вертикально вверх-вниз невозможно. И чем круче наклоняешь угол зрения, тем больше кривеет картинка. (Это помнят все, кто в своё время гонял Дюка).

Так, раз уж мы распрощались с акселераторами, пора, по моему разумению, достать этот скелет из пыльного шкафа, и вдохнуть в него новую жизнь:

Дюк, Doom:	А я сделаю вот как:
D (глубина) и Z (высота) координаты точки	D и Z координаты точки
⇩	⇩
$Y_{экранное}$	$Y_{вертикальное}$ (-17160..17160)
⇩	⇩
Точка помещается во Фрейм - буфер	Дли-и-инная таблица, покрывающая 89˚, значений этак на 100000. - для вычисления арктангенса
	⇩
	$Y_{экранное}$ (-300..300)
	⇩
	Точка помещается в Стрип - буфер
	⇩
	Сложнющее преобразование (1 раз за кадр)
	⇩
	Изображение перемещается во Фрейм - буфер (или сразу в видеопамять)



Минус всего этого дела очевиден: добавляется сложное преобразование всего, уже сформированного, изображения. Процессор греется от натуги, кэш-память переполняется, и показывает кукиш...

Плюсы менее очевидны, зато их больше:

- Резко упрощается преобразование координат из мировой системы в экранную, что позволяет нарисовать море вокселей. (а если ограничить вращение моделей

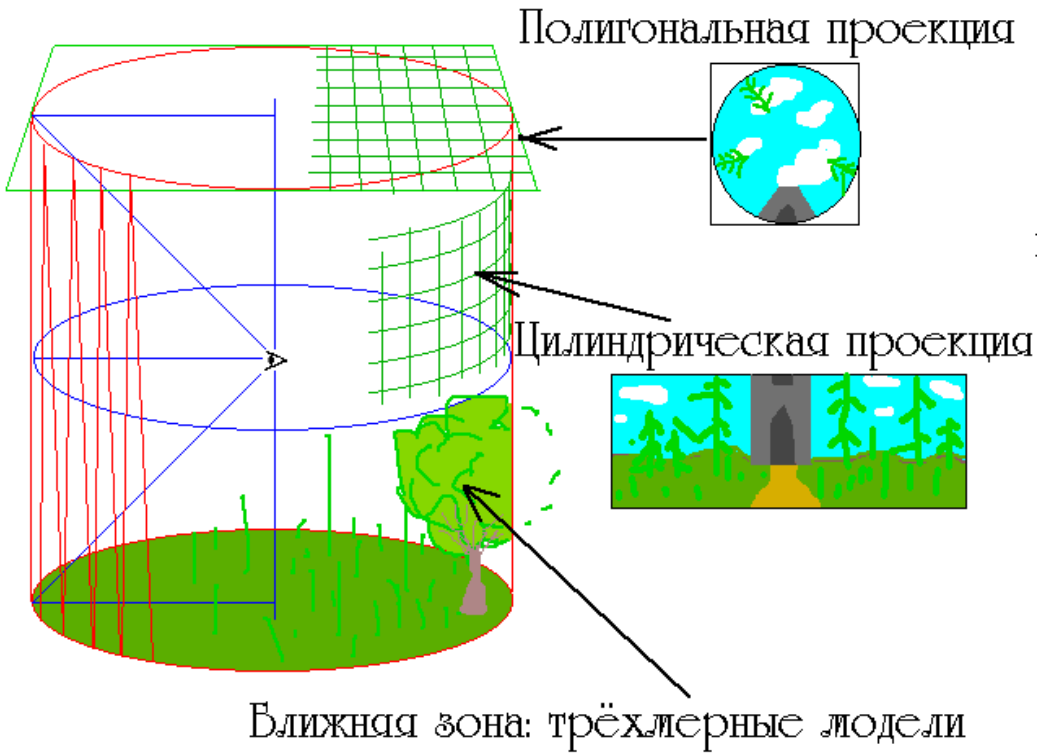
вертикальной осью - так это просто малина). Попутно появляется возможность, не прибегая к лишним затратам, применить Фонгову модель освещения.

-Резко упрощается рисование постоянных структур, типа зданий, с удалением невидимых комнат. Возвращается из небытия секторный механизм формирования уровней (а для AI-то это как удобно!)

- Если при преобразовании стрип-бфера во фрейм-буфер применить интерполяцию (bilinear filtering), то для разных участков экрана можно устанавливать разное разрешение: в центре - чётко, по краям - размыто. А экономия по быстродействию какая! А горы на горизонте как будут выглядеть - будто с anti aliasing'ом!..

- Проекция не обязана быть полигональной - её легко можно сделать хоть сферической, хоть яйцеидальной.

Кстати, этот метод можно применить и для 3d-ускорителей: использовать акселератор только для прорисовки ближней зоны, а основной процессор - для прорисовки дальней зоны, скармливая акселератору результаты его деятельности в виде двух текстур, натянутых на цилиндр:



А чтобы не расходовать зря ресурсы, стенки и крышку цилиндра обязательно надо делать дробными, с переменным разрешением: больше возле оси зрения, меньше

по краям. Будет достигнут тот же вышеописанный эффект экономии ресурсов и повышения чёткости в центре экрана.

Для этой техники, 16 битного Z-буфера будет совершенно достаточно, главное условие - каждый объект, пересекающий стенку цилиндра, обязательно рисовать обоими методами.

6. И последняя идея, довольно спорная. При наличии одного источника света (солнышко), создаём дополнительный буфер - глубины теней, и в нём ещё раз строим всю сцену, наплевав на текстуры и освещённость. Потом, рисуя сцену уже в нормальном буфере, вычисляем глубину и положение каждой точки в координатах буфера теней, и соответственно применяем освещённость - на солнышке / в тени. Естественно, буфер теней - дробный, с уменьшением разрешения по мере удаления от камеры.

В результате должен получиться эффект, как от ray tracing'a, но с меньшими затратами.

1.3. Расчёт системных требований

1.3.1. Минимальные системные требования.

В качестве платформы я выбрал Маздай 95, потому, что моя любимая DOS вымирает. Поддержка DOS режима в Маздае всё хуже, и старым программам приходится всё туже... Сначала были глюки с Sound Blaster, из-за которых замолчали все игры, распознавашие его автоматически. Теперь очередь дошла и до мыши, чувствительность которой параноидально мала, и не регулируется. И чтобы что-нибудь нарисовать, мне приходится перезагружаться в режиме полной эмуляции MS-DOS, иначе мой любимый De Luxe Paint II просто неработоспособен. Я уже не говорю про Arkanoid, который глючит сам, и завешивает систему намертво (Яркий пример, да ?..).

А в скором времени халтурщики из Майкрософт могут решить, что ради всякого старья не стоит и возиться, и вообще уберут поддержку DOS, чтоб какая-нибудь там Windows 2005 работала надёжнее.

Итак, с платформой вопрос ясен - Маздай 95. (Другие ОС я даже не рассматриваю, они стоят на 3% компьютеров, и если хотят жить, поддерживают Win32.)

Что же насчёт аппаратной части ?.. Минимум, на чём работает Windows (95-я, а не 98-я!) - это 486 с 8Мб ОЗУ. (Знаю, сам два года прожил в такой конфигурации) Но на 486 машине ничего путного не сделаешь. Самый крутой мотор, который тянет этот процессор - это движок Дюка, да и тот иногда притормаживает. К тому же, львиная доля 486 машин снабжена видеокартами ISA, у которых скорость обращения к видеопамяти редко превышает 1 Мбайт/с.

С другой стороны, сейчас на толкучке полно относительно дешёвых Пентиумов, и только совсем бедный не обзавёлся хоть завалящим. Весь вопрос - какую частоту принять за минимальную ?.. 75 и 100 уже сейчас выглядят несерьёзно (и встречаются редко), так что как рабочий вариант я принимаю 166. Эта цифра может измениться в процессе разработки, которая может занять годы. Учитывая скорость технического прогресса, я также принимаю обязательность поддрежки процессором технологии MMX, заметно упрощающей и ускоряющей некоторые алгоритмы.

Что касается памяти: 8 Мбайт - это так мало, что даже говорить не стоит.

16 Мбайт - вопрос спорный, удастся ли втиснуть и графику, и сложный игровой мир в это прокрустово ложе. В любом случае, получится уродливо, но как я уже говорил выше, в хорошей игре графика - не главное.

Не буду рассматривать тех недальновидных людей, кто поставил Windows 98 на машину с 16 Мб памяти, пусть апградят до 24.

Сначала имеем 16 Мбайт.

Windows 95 отъедает 3 Мб - остаётся 13.

Программа, откомпилированная в Дельфи, отъедает 1Мб - остаётся 12.

1 Мб оставлю в резерве - на всякий случай. Может, для кэша пригодится.

1 Мб уйдёт на разные служебные таблицы - остаётся 10. Их делим на четыре части:

Игровой мир, графическое ядро, модели и текстуры, звук. Прежде, чем делить, надо приблизительно оценить, какая часть от нормы приходится на каждую из категорий в таких экстремальных условиях, и насколько интенсивной будет подкачка недостающих частей.

Игровой мир - можно ужать достаточно серьёзно, нарубив мир клеточками, и просчитывая только те клеточки, где находятся персонажи. Остальные клеточки или заморожены (и пшли вон из памяти), или функционируют по упрощённой форме, влезающей в один килобайт (пример: лесной пожар. Для каждого дерева 8 байт: x, y, z, h, порода дерева, состояние, вероятность возгорания, интенсивность горения). Детально нужно иметь только визуальное представление ближайших к игроку клеток. Учитывая, что нужно хранить в памяти только физическое описание ландшафта, зданий и моделей, которое может быть довольно простым, и что размер клеточки можно подбирать в широких пределах, можно очень приближённо оценить мир в 2 Мб.

Графическое ядро - я имею в виду различные буферы, в которых формируется изображение. Экранный пиксел: 32bit Z+32bit pixel color+ 32 bit sfx = 12 байт. Умножив на 75000, получим около мегабайта. + 1Мб буферных спрайтов (пониженного разрешения), которые на хилой машине просто обязательны. Добавим различные предвычисленные таблицы, и получим 2,5 Мбайт.

Оставшиеся 5,5 Мбайт делятся между текстурами, моделями и звуками по ходу действия: текстуры - восьмибитные, звуки - 11кГц, модели - упрощённые.

Итак, минимальные системные требования:

Windows 95

Pentium MMX 133 MHz

16 Mb RAM

PCI video card with 640x480x16bit mode supported.

1.3.2. Рекомендуемая конфигурация.

Как показывает опыт, лучше всего удаётся приспособить игру к своей собственной машине, у коей в данном случае процессор AMD K6-2 с PR ≈ 400 (разогнан на 30%), и обычная 66 МГц мать. А вот требуемый объём памяти лучше изучить посредством расчёта.

Игровой мир: очень трудно оценить, не начав работать над ним. Разумным пределом были бы 5..10 Мбайт, учитывая объём сохранённой игры.

Графическое ядро: видеобуфер для 400x300 составит 1,5 Мб. Плюс буферные спрайты: 3Мб восьмибитные, 6Мб шестнадцатибитные. Плюс 0,5 Мб - предвычисленные таблицы. Получаем 5..9 Мбайт при разном уровне детализации.

Текстуры и модели: недаром у современных 3d акселераторов память меньше 16 Мб не бывает. Разнообраизия при среднем качестве можно достичь при наличии порядка 50 текстур - это 5..25 Мбайт при разном уровне детализации.

Звук: 2..3 ambient sounds, и пара сотен событийных (0.5..2 с.) при 22 кГц займут 6..10 Мбайт.

Суммируем: 7 Мб Windows 98 + 2Мб резерв 1Мб программа + 1Мб таблицы + 4..10Мб мир + 5..9 Мб ядро + 8..24 Мб текстуры + 4..10 Мб звуки = 32..64 Мб, при разном уровне детализации.

Итак, рекомендуемая конфигурация:

Windows 95/98

Pentium II MMX 366 MHz

64Mb RAM

??? а до аппаратного ускорения я ещё не дошёл.

1.4. Управление памятью.

1.4.1. Обзор положения

Если задаться вопросом, почему множество современных игр, в целом неплохих, так чудовищно тормозит, если им хоть чуть-чуть не хватает памяти, и если хорошенько исследовать этот предмет, то мы вытащим за ушко на солнышко огромную бяку, заботливо подложенную дядей Гейтсом на пути прогресса.

Я имею в виду систему управления виртуальной памятью, свойственную Windows98, и её чудовищное отставание по скорости от аналогичной части Windows95. Не знаю, чем руководствовались майкрософтовские бракоделы, создавая этот шедевр. Может быть, это была плата за устойчивость системы. Может быть, они применили к этому делу какой нибуть очень научный подход. А может быть, они настолько бесятся с жиру, что уже забыли про существование компьютеров медленнее третьего пентиума, и соответственных винчестеров...

По крайней мере, налицо результат: применение тестовых программ на моём компьютере (HDD ≈ 2500 Кбайт в секунду, время доступа 12 мс) показывает, что максимальная скорость подкачки страниц виртуальной памяти у Windows98 составляет около 270 Кбайт в секунду. (для сравнения, у Windows95 - 700..800 Кбайт в секунду при записи, и почти 2 тыс. при чтении) Причём, если производить только чтение из памяти, то скорость подкачки не возрастает, как должно быть, по идее.

А наивная игра принимает эту память за настоящую, и загружает в неё мегабайт этак семьдесят - вызывая пятиминутный обмен с диском, при котором информация (на 10% скорости) перекачивается с винчестера, на котором она есть в базе данных игры, на тот же винчестер, в файл подкачки Windows - и всё для того, чтобы быть снова прочитанной, когда понадобится... Бред! И кофе на эти "обеденные перерывы" не напасёшься.

Можно подумать, что создатели игр страдают тихим маразмом, но скорее всего, они давным давно не видали компьютера с памятью меньше 128 Мбайт, и о таких приземлённых вещах просто не думают, как не думают и об обладателях 16мегабайтных Пентиумов-166: раз у человека нет денег на апгрейд такого старья, то на лицензионную игру у него денег нет и подавно, и нечего стараться приспосабливать

её к дряхлым машинам. Короче, "без бумажки ты - букашка, а с бумажкой - человек". (я имею в виду зелёную такую, с хмурым мужиком посередине)

Так какой же должна быть действительно хорошая игра ?..

Во-первых, она должна быть нечувствительной к объёму памяти, и грузиться не дольше десяти - двадцати секунд. Этого можно достигнуть только отказавшись от использования виртуальной памяти Windows - не совсем, конечно, а только для базы данных. Программа должна знать объём физической памяти, действительно доступной ей, и отводить под базу данных не больше, чем остаётся после размещения кода, стека, всех данных и буферов. И, конечно, следить, чтобы всё это хозяйство оставалось в памяти, попадая на диск только при переключении пользователя на другую задачу.

Во-вторых, должна регулироваться в очень широких пределах детализация графики - основного потребителя ресурсов. Причём, желательно, не менее, чем в 10 раз. (от 16 Мбайт до 128, от сотого Пентиума, до пятисотой "Тройки") - выполнения этого правила добиться гораздо легче, чем предыдущего. Одно изменение режима дисплея даёт уже четырёхкратную разницу, да изменение алгоритма рисования быстро / красиво - ещё раза в два-три.

В-третьих, игра должна быть живучей, и не отбрасывать копыта после первой же ошибки доступа к памяти, или глюка в драйверах Direct Draw.

1.4.2. Реализация.

Гадость первая: в API Windows нет такой функции, которая позволяла бы однозначно определить, где находится страница ВП - в памяти, или на диске. VirtualLock пока - простая "затычка", не играющая никакой полезной роли. Но сама радость даже не в этом, а в том, что (по крайней мере, в Win98) GlobalLock - тоже фуфло!

Доказательство: у меня 32Мб ОЗУ. Распределяю по GlobalAlloc блок в 33Мб, затем вызываю GlobalLock его. Что получаем ?.. Правильно, указатель. Причём, ненулевой - т.е., врёт, зараза, ничего она на самом деле не заблокировала.

Гадость вторая: функция, возвращающая объём свободной физической памяти, имеет в виду память, не используемую ни одной программой (т.е., чаще всего, ноль), и ничего не говорит о том, сколько её можно спихнуть на диск, а сколько - намертво захапала система. Ну, а гадать здесь - дело опасное: у Win95 и Win98 весьма различные

аппетиты (3 и 7 Мб соответственно), а ишшо различное количество DLL-ов может быть загружено, и Win2000 нехорошо нависает над перспективой...

Обход этого всего иначе, как криво, реализовать не могу. Но криво - это, всё-таки, лучше, чем ничего. Дело несколько упрощает тот факт, что проектируемая игра априори не пойдёт на 486 машине, а значит, может беззастенчиво использовать Time Stamp Counter Пентиума. Т.е., измерять временные интервалы с точностью до микросекунд.

Итак, (модуль c_sys): сначала статистическим образом вычисляется минимальное время время копирования одной страницы памяти в другую. Затем, (используя предположение, что скорость подкачки не может быть более одной десятой скорости обращения к памяти, и что скорость подкачки не может превысить 400Мб в секунду) определяется среднее время "долгого" копирования страницы ($t>10*t_{min}$). Всё вышеприведённое измерение занимает много времени, так что измеренные параметры запоминаются для будущего использования. Попутно, конечно, определяется объём памяти и адреса, уже распределённые программой под код, стек, данные и пр. - используя VirtualQuery.

Затем, отталкиваясь от значения, полученного в последний раз, оценивается объём доступной памяти, методом многократного чтения из распределённого объёма. Причём, подкачка страницы распознаётся только по времени копирования ($t>0.1*t_{ср.долг.}$). Если объём доступной памяти сильно отличается от имевшегося в предыдущем сеансе, заново производится измерение средней скорости подкачки.

Вся эта ахинея, по идее, должна быть достаточно устойчивой, если трижды сплюнуть через левое плечо, и надеяться, что какой-нибудь "гений" из Microsoft не додумается включить подкачку в процесс кэширования.

1.5. Видеосистема.

Проект, в целом, включает freeware - библиотеку DelphiX от Hiroyuki Hori, обеспечивающую в весьма культурном виде поддержку DirectX от 5.0 и выше. В DelphiX все используемые dll'ы подключаются только динамически, благодаря чему программа пойдёт на машине без DirectX. Однако, объект TDXDraw должен быть помещён на форму визуальным методом, поэтому я для удобства и безопасности предпринимаю следующее:

Приложение имеет два окна, причём одно из них - главное, создаётся при инициализации программы, и не содержит ни одного визуального компонента. Второе окно - содержит все элементы DelphiX для DirectDraw, DirectInput, и пр. Оно создаётся по окончании инициализации, и переключении в требуемый видеорежим - конечно, только при условии наличия DirectX. Это окно уничтожается каждый раз, когда приложение теряет фокус. А потом создаётся заново.

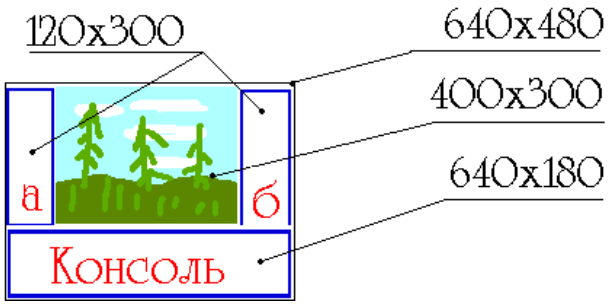
Видеосистема предоставляет программе абстрактные процедуры доступа к фреймбуферу (который может быть как сурфейсом DirectDraw, так и потрохами битмапы), и смены буфера (DDraw.Surface.Flip, либо же вывод битмапы в главное окно через GUI)

Шрифт поддерживается только встроенный, устраняя зависимость от наличия или отсутствия в системе русских шрифтов, а главное - от неопределённой ширины букв в них. Система поддержки диалогов (модуль c_mernu) запускается до выдачи запроса о выборе языка (как, собственно, и двузадачная поддержка, поначалу работающая от системных часов).

Как уже упоминалось выше, толстые рамочки и дополнительные окошки - негуманный, но эффективный способ сократить объём вычислений. К тому же, это решает одну заковыристую проблему: Для PR (166..400) системы, оптимальным размером 3d-окна являются от 320x240 до 400x300. Но такие видеорежимы поддерживает очень малое число видеоадаптеров, особенно режим 400x300. То есть, большинству пользователей для получения окна в 400x300 придётся переключиться в режим 640x480, окно займёт середину экрана, а края придётся закрашивать каким-то

фоном. Так не лучше ли занять это место полезной информацией, которой в сложной игре всё равно должно быть много?..

Окончательно, принимаю:



Консоль удобна для размещения нескольких строк текста - для организации меню, вывода пространных сообщений, или ведения диалога с NPC. Для получения полноэкранных меню или заставок, она просто раздвигается на весь экран.

Окна а и б (их можно соединять в одно по ходу действия, двигая 3d-окно) больше подходят для размещения различных индикаторов - проценты здоровья, ощущение среды, инвентарь и пр.

Так как в проектируемой системе кнопки мыши и клавиши клавиатуры будут равноправны, то клавиатуру можно рассматривать как приставку к мыши, или, другими словами, мы имеем стокнопочную мышь. Это позволяет в каждом окне иметь собственный мышиный курсор, функционально связанный с определённой клавишей.

Плюс очевиден: можно одновременно работать в нескольких меню, не таская курсор через весь экран, и не переключаясь между окнами. (Пример: прицеливаемся в оппонента, затем в окне б - мышью - выбираем оружие, при этом лишь слегка сдвинув прицел)

Возможные режимы экрана для видеосистемы:

а). Уменьшенный 64%-ый. (Многие адаптеры в последнее время поддерживают видеорежим 512x384, и большинство мониторов позволит растянуть аналогичных размеров окно на весь экран в режиме 640x480)

- экран 512x384 (197тыс.пикс.), окно 320x240(77тыс.пикс.)

б). Нормальный 100%-ый. - экран 640x480 (307т.п.), окно 400x300 (120т.п.)

в.) Увеличенный 156%-ый. (для машин на Pentium III) - экран 800x600 (480т.п.), окно 500x375 (188т.п.)

г.) Увеличенный 256%-ый. (для маньяков с 17..21д. мониторами)

- экран 1024x768 (786т.п.), окно 640x480 (307т.п.)

1.6. Распределитель времени.

Поскольку в любой игре львиная доля времени процессора тратится на работу с графикой, нельзя рассматривать подсистему распределения времени в отрыве от графического ядра. Наоборот, они должны составлять единое и неразрывное целое, кстати, вместе с менеджером базы данных. Поэтому, разработку распределителя времени я начну с краткого анализа графического ядра. Расчёт лучше вести для наихудшего случая (Пентиум 166 с 16 Мб памяти).

Скорость чтения/записи ОЗУ (при неэффективном кэшировании, какое обычно случается при переносе больших массивов данных) составляет 66 млн. машинных слов в секунду, или 266 Мбайт/с. В то же время, для средней PCI видеокарты, скорость записи в видеопамять составляет 33 Мбайт/с, то есть лимитирующим фактором является именно объём данных, переносимых в видеопамять.

Время заполнения окна:

- 320x240x16 - 4,5 мс, или 12% кадра.
- 400x300x16 - 7 мс, или 19% кадра.
- 640x480x16 - 18 мс, или 50%(!) кадра - Ещё одно доказательство, что для обычного Пентиума (PR=166..400 МГц) этот размер экрана является непосильным бременем. Даже такой простой и эффективный восьмибитный мотор, как Quake II Software Mode, на машине с PR400 тянет всего 5-10 FPS .

За базовый вариант для проектируемой системы принят режим 640x480 с трёхмерным окном 400x300, которое обновляется каждый кадр. Консоль и боковые окна должны быть статичны, то есть не содержать интенсивно анимированных участков, тогда соответствующие им области видеопамяти будут обновляться только при изменнии содержимого этих окон, которое нецелесообразно производить чаще 5 раз в секунду.

Среднее результирующее время заполнения видеопамяти при этом составит не более 10мс, или 25% кадра, максимальное - не более 50% кадра.

Итак, обратно к распределителю времени:

Поскольку качественно реализовать многозадачный режим под Windows практически невозможно, реализую это так:

Основной алгоритм, начиная от первичной инициализации, выполняется почти линейно, от процедуры к процедуре, внутри которых с разумной щедростью рассыпаны процедуры перехода (CYC), с таким расчётом, чтобы при наихудших условиях (Pentium 166) выполнялись не реже, чем каждые пять - десять тысяч микросекунд.

Внутри процедуры CYC проверяются условия переключения, и производится вызов циклической процедуры с заданной частотой.

Вывод из фреймбуфера в видеопамять, перемещение мышиных курсоров, подсветка кнопок меню (контурами), восприятие управления, счётный цикл игры - выполняются в циклической процедуре.

Все остальные операции - включая загрузку, управление меню, синтез трёхмерного изображения - выполняются обычным порядком.

Дополнительные условия:

1) В циклической процедуре вообще не производится обращений к базе данных, чтобы минимизировать обмен с диском. Для этого физическое описание игрового мира составляется линейной процедурой загрузки мира, и запирается в памяти.

2)

1.7. менеджер базы данных.

1.7.1. Общие положения.

В базе данных, в общем случае, хранятся два вида записей: большие, чем страница, и меньшие, чем страница. Проблема в том, как обращаться с сонмом записей размером в несколько сот байт: считать каждую самостоятельным объектом - значит при загрузке угробить уйму времени на перевод головок винчестера. Кучковать мелкие записи в группы - значит угробить уйму памяти впустую, что приведёт к её нехватке с последующей интенсивной подгрузкой.

Оценим проблему в числах:

Время чтения, всего, мс (чтение, %)	страницы в 4Кбайт	страницы в 8 Кбайт	страницы в 16 Кбайт	страницы в 32 Кбайт	страницы в 64 Кбайт
Мой винт: послед.обр. 3 Мб/с, поиск 12мс	13,3 (10%)	14,7 (18%)	17,3 (30%)	22 (45%)	33 (65%)
Хороший винт: послед.обр 10 Мб/с, поиск 4 мс.	4,4 (10%)	4,8	5,6	7,2	10,4 (61%)

Может быть, цифры слегка притянуты за уши, но отсюда хорошо видны причины крайне низкого быстродействия виртуальной памяти Windows: страница в 4Кбайт черезмерно мала. Для повышения эффективности подкачки, размер страницы должен составлять 64..256 Кбайт, больше - нецелесообразно, так как диск может быть фрагментирован, да и время чтения значительно возрастёт.

Для повышения же эффективности использования памяти, мелкие записи нужно сортировать по типу, затем по размеру, затем по названиям - больше вероятность, что в одну страницу попадут "родственные души", которые понадобятся одновременно Также, мелкие записи можно удалять из памяти не постранично, а поштучно, и перемещать остающиеся из полупустых страниц в почти полные - то есть, дефрагментировать память.

Сжатие данных позволяет уменьшить время чтения страницы, но не время поиска. То есть, при его применении, вышеприведённый размер страницы относится к *ужатой* странице. Реальный же её размер (разный для всех страниц) может быть много больше, принимая, что скорость декомпрессии значительно превосходит скорость

чтения. Отсюда, большое значение имеют способы сжатия данных внутри страницы - каждая запись отдельно, страница целиком, либо различия записей.

Чтобы исключить неприятности с перемещением находящихся в работе данных, и чтобы повысить эффективность подкачки, следует во всех разделах программы придерживаться разделённого алгоритма:

1). Составляется список операций, и необходимых им данных.

2). Производится запрос в базу даных.

3). Выполняются операции по списку.

При таком подходе, производится подкачка сразу массива страниц, в свою очередь с составлением плана подкачки и его оптимизацией. (Например, при малых расстояниях между страницами, вместо перескока на новую позицию может применяться чтение ненужных данных, которое пройдёт быстрее, чем перевод головок винчестера.)

Алгоритм оптимизации подкачки должен меняться в зависимости от объёма доступной памяти: При её нехватке, из прочитанной страницы к окончательному "месту жительства" должны переселяться только записи, указанные в списке подкачки. При избытке памяти, в ней можно размещать всю страницу целиком, в том числе и непрошенные записи, которые могут потребоваться в дальнейшем.

1.7.2. Реализация.

1.7.3. Форматы представления информации.

1.8. Порядок инициализации.

1). Разделы инициализации модулей (порядок может варьироваться):

а). c_video: запоминается размер экрана для восстановления при выходе (`c_video.WindowsDisplay: TDevMode`)

б). c_scary: грузится функция "RegisterServiceProcess" из библиотеки kernel32.dll

2). WEXTENDE.InitWext:

а). Проверяется способность процессора выполнять команду RDTSC, и выход с руганием при неспособности.

б). Проверяется способность процессора выполнять MMX - команды, и выход с руганием при неспособности.

в). Создаётся главное окно приложения - с ожиданием, пока оно действительно будет открыто.

г). Устанавливаются обработчики событий приложения - получение/потеря фокуса.

д). Включается модуль работы с системным реестром.

е). Инициализируется видеосистема, включая попытку создания второго окна с перечислением доступных режимов DirectDraw, и проверкой наличия DirectX. Устанавливается использовавшийся в предыдущем сеансе видеорежим (записывается в реестр).

ж). c_MK.InitMK: подключение библиотеки c_hook.dll с выковыриванием из неё функций управления, и выход с руганием при отсутствии оной, или невозможностии перехвата системных цепочек сообщений мыши/клавиатуры.

з). Включается система управления меню.

🡻**** *начиная с этого момента, возможен диалог с пользователем* ****🡻

и). Устанавливается рабочий язык, и при отсутствии отметки в реестре выводится запрос о выборе языка.

й) Производится запоздалое ругание при отсутствии DirectX.

к). Тщательно измеряется тактовая частота процессора, с запоминанием в реестре и предупреждением при изменении с последнего сеанса, и ругание при недостаточности оной.

л). c_sys.Init:

- определяется версия ОС, и ругание, если она не 95/98/NT.
- загружаются из реестра статистические параметры
- зачищается память, и при необходимости производится статистический анализ быстродействия.

м). c_db.Init:

- составляется список файлов БД

н).

2. Редактор трёхмерных объектов.

2.1. Формат представления информации.

Поскольку модели строятся по принципам, сходным с ООП, хранить их на диске следует в текстовых файлах, содержащих специальные команды наследования и связывания.

2.2. Основные понятия, и реализация.

3. Форматы представления информации.

(Собраны совершенно бессистемно, в порядке разработки.)

3.1. Шрифты.

Оные зашиты в программу в виде констант в модулях C_SysFNT и C_HdlFNT.

Системный шрифт - (3..8x7), и его увеличенно - улучшенная копия (6..16x14):

```
Const SysF7: array[' '..#255,0..8] of byte = (...
```

`SysF7 [<буква>, x]`, где $x \leq 7$:
бит 7 всегда = 0, биты 0 .. 6 - верхняя .. нижняя точки (1 - есть, 0 - нет)

`SysF7 [<буква>, 8]` - ширина символа, 0..8.

```
Const SysF14: array[' '..#255,0..15] of word = (...
```

`SysF14 [<буква>, x]`, где $x \leq 13$:
биты 14 и 15 всегда = 0, биты 0 .. 13 - верхняя .. нижняя точки (1 - есть, 0 - нет).

Ширина определяется, как `SysF7[<буква>, 8]*2`.

Красивые заголовочные шрифты определяются, как массив 64битных значений (Int64).

В наличии пока 3: Bard - из DPII, Old - "Cyrillic Old", Brush. - "Technical Di"

сначала следуют 224 30-элементных записи формата:
0 - индекс начала описания буквы в массиве.
1 - ширина буквы
2..29 - 224 знаковых байта, описывающих зазор от предыдущей буквы, если она указанного типа.
Итого, 53Кбайт - один "заголовок" шрифта.

Далее следует массив, представляющий сами буквы, 64 пиксела высотой. Младший бит - верхний пиксел.

3.2. Формат файла базы данных.

3.3. Мир

4. Игровой мир

Заключение

Рановато, пока что, заключение писать.

1234567890=` []{};:'",.<>\|/!@#$ %^&*()abcdefghijklmnopqrstuvwx yzABCDEFGHIJKLMNOPQRSTUVWX YZабвгдеёжзиёклмнопрстуфхцч шщъыьэюяАБВГДЕЁЖЗИЁКЛМНО ПРСТУФХЦЧШЩЪЫЬЭЮЯ

Приложения

А. Показатели быстродействия для различных компьютеров

1). Время выполнения команд в тактах процессора:

	P166 (2,5:1)	AMD K6-2, 350 (5:1)	PII	PIII
Арифметические- 32, с независимыми операндами		50%		
Арифметические -32, с 100% зависимыми операндами		80%		
Арифметические- 16, с независимыми операндами		100%		
Арифметические -16, с 100% зависимыми операндами		100%		
Арифметические -8, с независимыми операндами		90%		
Арифм. 8 + арифм. 32 с тем же регистром (суммарно)		400% (предш.1); 500% (предш.2)		
Арифметические -32, (РЕГ, ОЗУ)		400% (0,8 $\tau_{мп}$)		
то же, (ОЗУ, РЕГ)		600% (1,2 $\tau_{мп}$)		
CMP		100%		
Jxx, переход		100%		
Jxx, нет перехода		100%		
Умножение-32 со знаком		290%		
то же, из памяти		523%		
Умножение-32 без знака		290%		
Умножение -16 без знака		300%		
Деление-32 со знаком (в потоке арифм. команд)		**2200%**		
MMX - умножение (нулей)		180%		
MMX - умножение (значений)		190%		

Примечания