№19
2012

Независимый электронный журнал о разработке компьютерных игр
Издается с 2008 г. Доступен по CC-BY-NC-SA

FPS

Путь Программиста
обретение дзен...

Да будет... звук!
D, Linux и OSS

Freestyle:
первое знакомство

Веб-сервер «своими руками»
Часть II

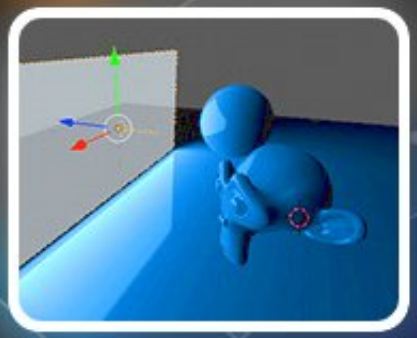
Интервью с Колином Леви

+ многое другое!

Главный редактор: Тимур Гафаров
Дизайн и верстка: Тимур Гафаров

fpsmag.zymichost.com

Blender



Новости из мира Blender
Freestyle. Первое знакомство
Интервью с Колином Леви

Путь Программиста

Полезные команды

Язык D

Новости
Шаблоны. Часть III
Да будет... звук!
Веб-сервер своими руками. Часть II

Python

Гвидо ван Россум
о перспективах развития языка

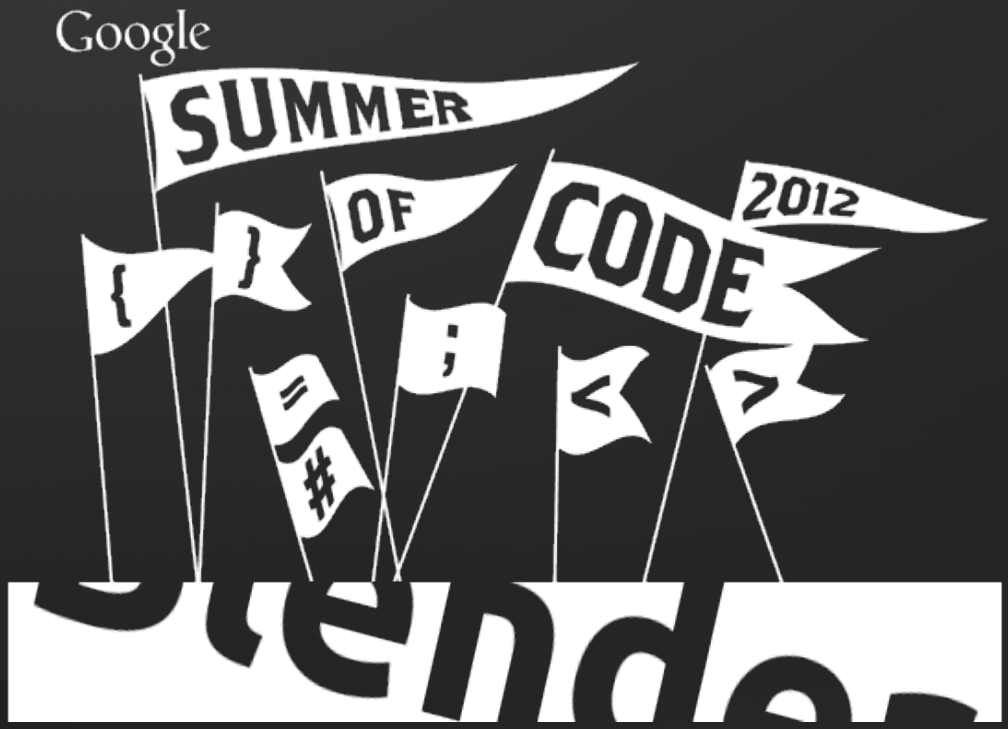
Blender: Новости

В конце апреля вышла новая версия Blender 2.63. Помимо дебюта небезызвестной системы полигонов BMesh, о котором мы уже упомниали в прошлых номерах журнала, в данной версии значительно обновлен движок рендеринга Cycles: добавлена поддержка фоновых изображений при предварительном просмотре, Ambient Occlusion, слоев-масок для получения прозрачных областей на изображении, а также HDR и текстур в формате с плавающей точкой (float precision). Добавлена возможность скрытия частей модели в режиме лепки для увеличения производительности и доступа к частям меша, которые в ином случае труднодоступны. Кроме того, были улучшены инструменты Motion Tracking, внесено множество других доработок и исправлений.

В этом году Blender Foundation традиционно участвует в Google Summer of Code – инициативной программе, в рамках которой проводится отбор открытых проектов, в которых могут принять участие студенты. Победителям выплачиваются денежные гранты. Проекты должны предложить будущим участникам задания по программированию в рамках проекта. Каждый проект получает $ 5500, из которых 5000 достается студенту-участнику, выполнившему задание, а 500 — самому проекту. В случае с Blender, участник должен либо улучшить существующий код, либо реализовать новую возможность по запросу сообщества. Например, внести улучшения в новый рендер Cycles, систему BMesh, игровой движок, инструмент Retopology, поддержку формата Collada, режим лепки, редактор UV-развертки и т.д.

С подробным списком идей можно ознакомиться на странице http://wiki.blender.org/index.php/Dev:Ref/GoogleSummerOfCode/2012/Ideas

Информация для участников GSoC '12:
http://www.google-melange.com/gsoc/org/google/gsoc2012/blender
Руководство для начинающих разработчиков Blender:
http://wiki.blender.org/index.php/BlenderDev/New_Dev_Info



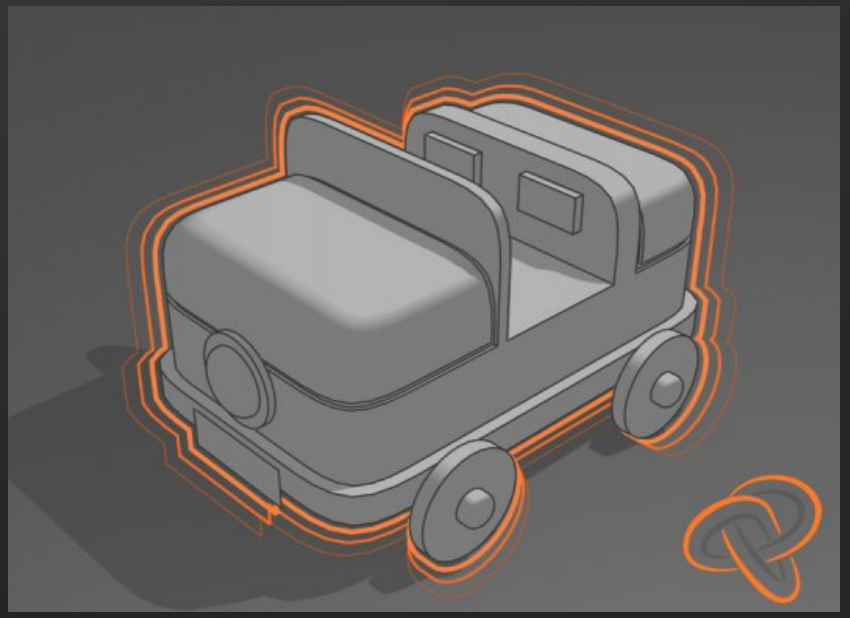
Съемочная группа проекта Mango завершила работу над кик-оффом Quit Blender – это короткометражный шутливый фильм, в котором члены команды играют самих себя в «роковой день» – когда созданные ими же в Blender роботы и другие фантастические персонажи из виртуального мира «оживают» и пытаются напасть на своих создателей...

Проведенная работа отлично иллюстрирует основную цель проекта – улучшение VFX, отслеживания движений (motion tracking), движка рендеринга Cycles и других новинкок Blender последних лет.

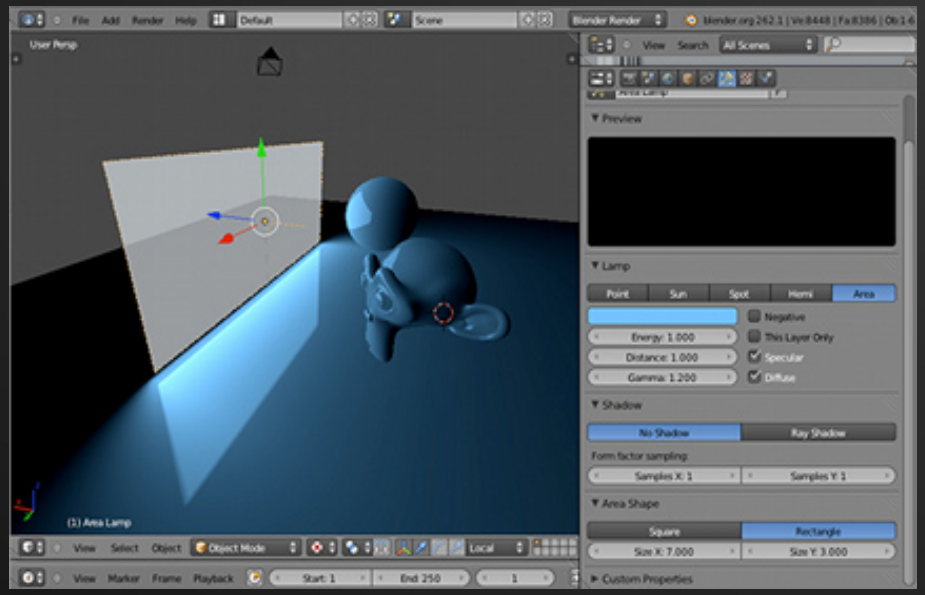
Фильм доступен на YouTube в различных разрешениях, включая Full HD (1080p). Все модели и натурные съемки доступны по лицензии CC-BY.

Хорошая новость для любителей нефотореалистичного рендеринга – недавно поступили вести из блога интеграции Freestyle. Для тех, кто не в курсе: это один из проектов GSoC 2008, независимый NPR-рендер, разрабатываемый компанией Academic Research Group, интегрированный в Blender усилиями программистов Максима Куриони и Тамито Кадзиямы. Обновления в проекте включают новые возможности в редакторе параметров, улучшения импорта мешей и множество багфиксов.

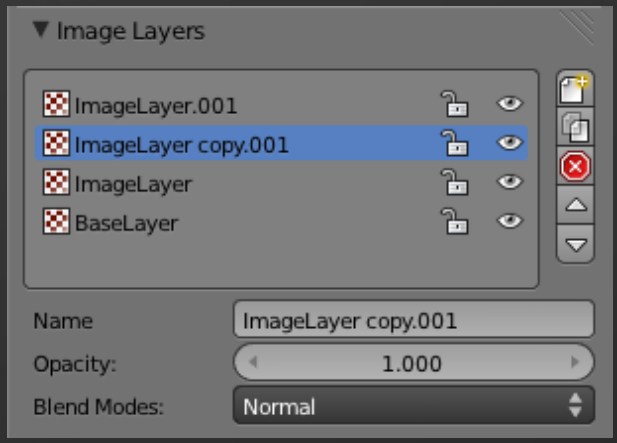
http://freestyle.sourceforge.net
http://freestyleintegration.wordpress.com



Не стоит на месте и развитие игрового движка Blender (BGE). Как стало известно, в рамках ветки Harmony движок теперь поддерживает подповерхностное рассеивание (sub-surface scattering, SSS) и зональные источники света (area lamps). В планах также – поддержка параллакс-маппинга, отражений и преломлений, а также композитных узлов для визуального создания эффектов постобработки без необходимости вручную писать шейдеры на GLSL. А основная цель проекта Harmony – улучшение ситуации со светом и тенями в BGE. Планируется реализация технологии Variance Shadow Maps (VSM), поддержка анти-алиасинга и базового HDR-рендеринга.



А разработчик Фабио Руссо недавно опубликовал результаты своей работы по улучшению интерфейса слоев в Blender. В редакции Руссо организация слоев больше напоминает традиционную «стопку», как в графических редакторах (Photoshop или GIMP), что оказалось очень востребованным в среде пользователей пакета.



Gecko
gecko0307@gmail.com

Freestyle: Первое знакомство

Привычная в компьютерной графике генерация фотореалистичного изображения, которое смотрится «точно так же, как в реальном мире», всем прекрасно знакома. В противоположность такой традиции, нефотореалистичный рендеринг (non-photorealistic rendering, NPR) – это любая методика, которая позволяет получать изображения моделируемой трехмерной сцены в каком-либо ином стиле, нежели привычный реализм. Часто эти стили напоминают живопись или различные виды технической и художественной графики – чертежи, эскизы, рисунки чернилами, гравюры и т. д. Алгоритмы NPR также могут быть направлены на имитацию классической рисованной мультипликации. Проект по интеграции NPR в Blender носит название Freestyle.

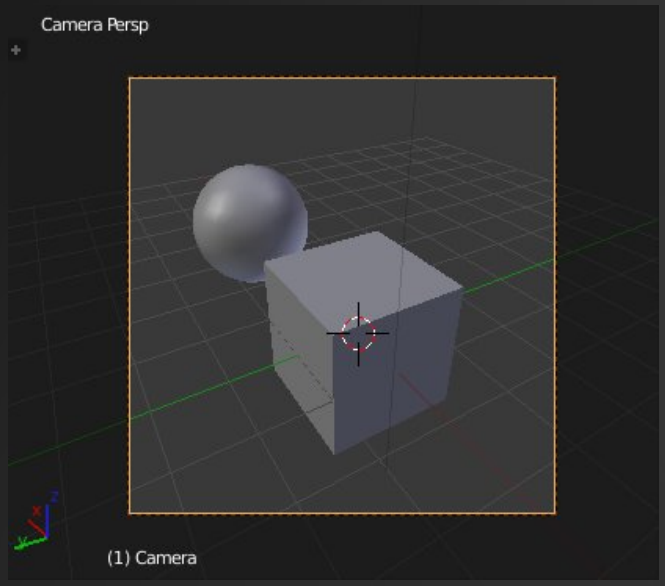
Freestyle – узкоспециализированный NPR-рендер, главная задача которого – отрисовка контуров на изображении. При этом гибкий интерфейс движка позволяет не только определить цвет и форму штрихов, но и задать политику распознавания линий. Например, можно не рисовать невидимые контуры. Или, наоборот, «обвести» нужные вам грани на моделях – например, глаза персонажа.

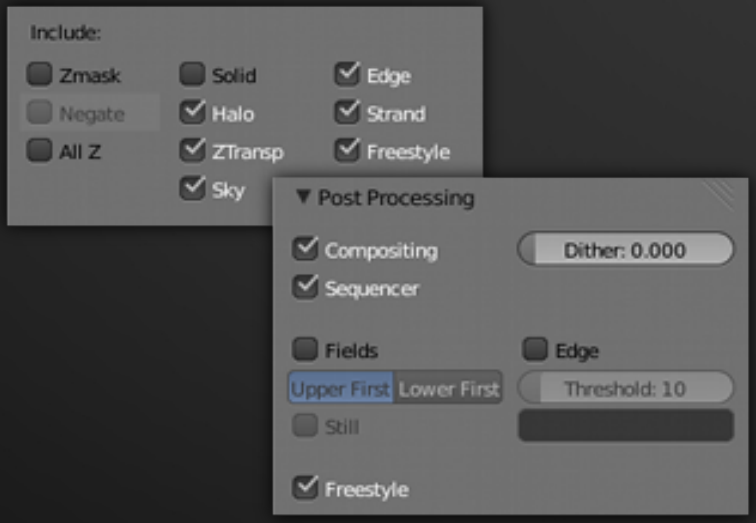
Freestyle разработан в виде гибкого программируемого интерфейса, который предоставляет максимальную свободу управления стилем: при помощи специальных операторов пользователь определяет собственный алгоритм того, как силуэты и контуры трехмерной визуализации должны быть преобразованы в стилизованные штрихи. Этот подход, во многом вдохновленный языками описания шейдеров из таких движков как, например, знаменитый RenderMan, решает проблему ограниченности интегрированного программного обеспечения, сводя все функции к нескольким базовым операциям и позволяя реализовывать на их основе самые сложные и разнообразные художественные стили.

Язык описания стилей Freestyle базируется на Python – в комплекте с движком поставляется много готовых стилей-скриптов. Если вы не умеете программировать, можете воспользоваться графическим интерфейсом (режим редактирования параметров – **Parameter Editor Mode**).

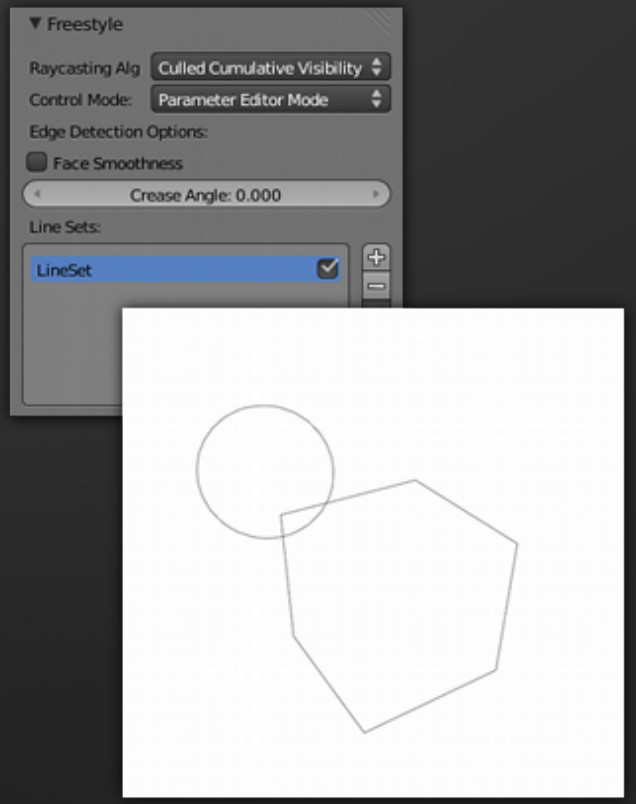
Найти свежую сборку Blender с включенным Freestyle можно на замечательном сервисе **GraphicAll.org**. У некоторых пользователей движок отказывается работать, ссылаясь на отсутствие нужных модулей Python. В этом случае, как правило, нужно перед запуском Blender записать в переменную среды PYTHONPATH путь к директории с модулями Freestyle. Для этого можно написать скрипт. У меня в Linux он выглядит примерно так:

```
WD=`dirname "$0"`
PROGRAM="blender"
PYTHONPATH=${WD}/2.62/scripts/freestyle/style_modules
export PYTHONPATH
"$WD/$PROGRAM"
```

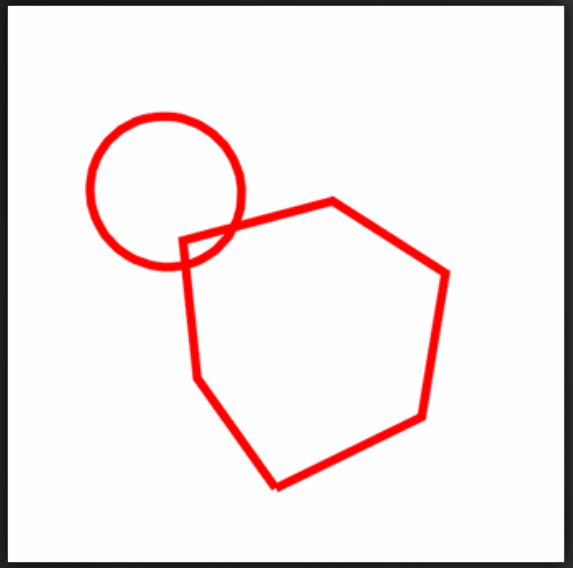


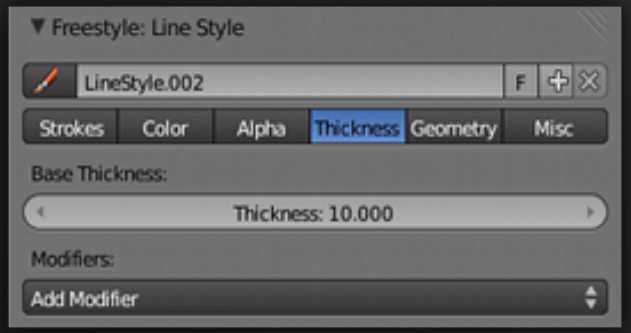


Создадим какую-нибудь несложную сцену для быстрой проверки работы Freestyle. Я сделал белый фон (**Horizon Color**) и на вкладке **Layers** в свойствах **Render** отключил рендеринг сплошных граней (**Solid**). Если у вас не стоит галочка напротив **Freestyle**, обязательно поставьте ее, иначе контуры не будут отрисованы. Кроме того, убедитесь, что во вкладке **Post Processing** также включена опция **Freestyle**. Теперь на вкладке **Freestyle** добавьте новый контур (**Line Set**) и попробуйте отрендерить.

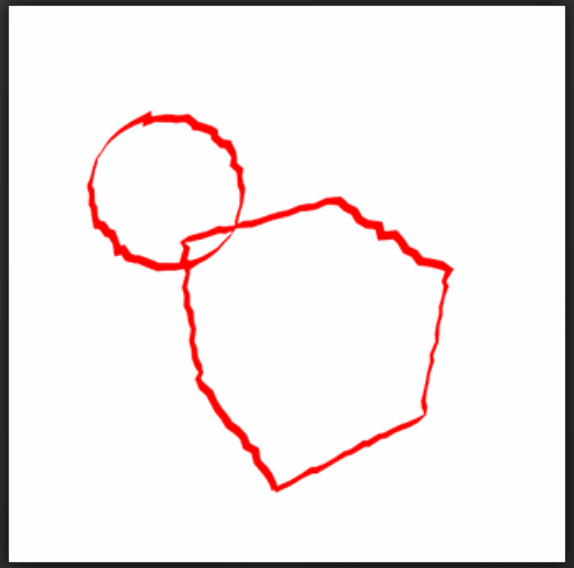


Можно увеличить толщину контура – на вкладке **Freestyle: Line Style** откройте панель **Thickness** и увеличьте параметр **Base Thickness** (толщина указывается в пикселях). На панели **Color** можно изменить цвет контура.





На любой из панелей свойств контура есть возможность добавить один или несколько модификаторов, которые процедурно изменяют то или иное свойство в зависимости от каких-либо величин.



Например, на той же панели **Thickness** есть модификатор «каллиграфия» (**Calligraphy**), а на панели Geometry вы найдете большой выбор модификаторов изменения направления линии – скажем, шум или линейное смещение. Это должно добавить рисунку элемент случайности, присущий традиционной графике, а также более реалистично сымитировать ту или иную художественную технику или инструмент.

Попробуем теперь вооружиться текстовым редактором и попробовать описать какой-нибудь несложный стиль в виде Python-скрипта:

```
from freestyle_init import *
from logical_operators import *
from PredicatesU1D import *
from PredicatesB1D import *
from Functions0D import *
from shaders import *
```

Выбираем только видимые грани:

```
Operators.select(QuantitativeInvisibilityUP1D(0))
```

Задаем топологию штрихов:

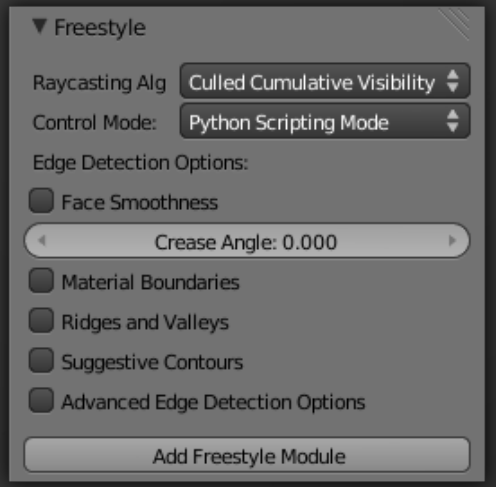
```
Operators.bidirectionalChain(ChainSilhouetteIterator(),
    NotUP1D(QuantitativeInvisibilityUP1D(0)))
```

Добавляем модификаторы-шейдеры:

```
shaders_list = [
    SamplingShader(50),
    IncreasingThicknessShader(2, 25),
    pyTVertexRemoverShader(),
    TipRemoverShader(3.0)
]

Operators.create(TrueUP1D(), shaders_list)
```

Переходите в режим **Python Scripting Mode** и добавьте этот скрипт.



Колин Леви: «Открытое ПО – это великая вещь!..»

Известный бразильский CG-ресурс **RPDesigner** недавно взял интервью у режиссера «Синтел» Колина Леви, который, как стало известно, перешел на работу не куда-нибудь, а в саму Pixar...

Привет, Колин! Для начала хочу сказать, что для меня честь брать у тебя интервью. Расскажи читателям немного о себе.

Для меня честь – давать интервью! Мне 23 года, я занимаюсь режиссурой и компьютерной графикой, живу в Уэст-Кост в США. Мне выпала большая удача воплотить свои мечты в жизнь, получив возможность работать в Pixar Animation Studios. Если вы хотите получить представление о некоторых моих работах, милости прошу на мой персональный сайт: **http://www.colinlevy.com**.

Признаюсь, я большой поклонник твоего творчества в качестве режиссера и аниматора. Когда именно оно пришло в твою жизнь и почему ты решил идти этим путем?

Спасибо! Это был постепенный процесс. Мне повезло вырасти в очень творческой семье. Мой дед – художник, тетя и дядя – актеры, а родители – музыканты. Когда мне было около 10 лет, я заинтересовался фотографией. Я любил фотографировать – это была моя первая настоящая страсть... Когда я открыл для себя видео через несколько лет, это было естественное развитие моего увлечения. Я просто следовал своим интересам – освоил навыки цифрового видеомонтажа, открыл возможности анимации, нашел Blender... Окончив школу, я принял решение продолжить эту работу профессионально.

Я все не могу перестать говорить о «Синтел». Это настолько впечатляющий фильм! И, без сомнения, важная веха в истории Blender. Не мог бы ты рассказать о своем опыте при съемках фильма?

Это был удивительный опыт, от начала до конца. Я очень счастлив, что получил возможность поучаствовать в этом проекте. Это была, конечно, чрезвычайно сложная задача. Я многое узнал о кино, о анимации, о совместной работе... Я вышел из этого проекта совершенно другим человеком! Один год – казалось бы, не так много, но, такое чувство, что съемки «Синтел» длились очень долго... Для меня было честью работать с такой талантливой группой художников, аниматоров и программистов. Мне очень понравилось работать над короткометражным фильмом – и я надеюсь, что это не в последний раз!

Твоя картина «В пути», которая завоевала множество наград, – тоже важный этап. Хочу отметить прекрасную музыку – она отлично подходит для короткометражного фильма. Расскажи подробнее об этом проекте.

Я начал работать над «В пути» раньше чем над «Синтел». Многое из области монтажа, спецэффектов и музыки для этого фильма было сделано параллельно с «Синтел». Это было очень непросто – выполнять обязанности по двум проектам сразу...

«В пути» – мой первый игровой фильм. У меня была большая команда студентов из Колледжа искусств и дизайна Саванны. Нам повезло, мы сумели заполучить необходимые ресурсы – дорогую технику, машины скорой помощи и т. д. – со сравнительно небольшим бюджетом. У меня было очень мало денег. Многое мы смогли сделать бесплатно.

Посмотреть «В пути» онлайн можно здесь: **http://enroute.colinlevy.com/film.html**.

Твой новый фильм «Тайное число» еще не закончен, но уже получил награды! Это вызывает такое чувство, что поклонники возлагают на тебя большие надежды. Не мог бы ты раскрыть какие-нибудь подробности о нем?

Ой, большие надежды заставляют меня нервничать... Я очень надеюсь, что все, кто принял участие в работе, остались довольны фильмом. Это был амбициозный проект. Я думаю, он получился очень хорошо, но есть многое, что я в нем просто ненавижу! Но то же самое можно сказать о любом из моих фильмов... Я вижу все свои недоработки – это отличный способ учиться.

Одна из вещей, которыми я горжусь в «Тайном числе» – это атмосфера фильма, которая была достигнута в первую очередь посредством звука и музыки. Команда по звуку была вовлечена в работу в течение почти целого года. И мне посчастливилось снова работать с Яном Моргенштерном, который также написал саундтрек к «Синтел» и двум другим открытым фильмам. Он очень талантливый композитор и вообще хороший парень. Он проделал большую работу в этом фильме.
Блог фильма: **http://secretnumber.colinlevy.com**.

Новость о том, что ты работаешь в Pixar, привело в восторг пользователей Blender по всему миру. Что это для тебя значит? Изменит ли это твою жизнь?

Честно говоря, я до сих пор не могу поверить, что это происходит на самом деле! Я в студии уже почти пять месяцев, и каждый раз, когда я вхожу в здание, я чувствую волнение. Pixar не совсем такая, как я себе представлял все эти годы, но это оказалось так же удивительно, как я и ожидал. Эта среда очень вдохновляет. Возможность работать в Pixar уже изменила мою жизнь. Я узнал многое о производстве крупных анимационных фильмов, и о том, каково быть частью такого сложного процесса. Я не знаю, что меня ждет в будущем, но сейчас я в восторге от своей работы.

Все знают, что ты используешь Blender. Как ты относишься к использованию открытого программного обеспечения в производстве фильмов?

Я всегда был большим поклонником Blender, и я по-прежнему провожу много времени, работая в нем в свободное время. Кроме того, я использую Blender для нескольких эпизодов с визуальными эффектами в «Тайном числе». Мне было очень интересно изучать все новые функции в версии 2.6. Я уже применил инструмент динамического закрашивания в кинопроизводстве – он работает очень здорово!

Blender быстро развивается. Я думаю, повсеместный переход на него – всего лишь вопрос времени. Оупенсорс играет важную роль в производстве крупных голливудских фильмов. Большинство рабочих станций уже основано на Linux. От операционной системы до языков описания шейдеров и физических движков – оупенсорс уже является «частью уравнения». Но я, если честно, знаю не так много, чтобы комментировать дальше. Я считаю, открытое ПО – это великая вещь. И, как доказывает Blender Foundation, открытые инструменты вполне годятся для профессиональной работы.

И последнее: что ты можешь посоветовать тем, кто хочет следовать тем же путем, что и ты?

Я знаю, что мне очень повезло. Я, конечно, упорно работал, но я прекрасно понимаю, что без везения тут не обойтись. Таким образом, я могу сказать лишь одно: занимайтесь тем, чем вы увлечены, и всегда стремитесь улучшить свою работу – вот и все. Да – и работайте постоянно! Пусть хотя бы 15 минут в день – важно добиться прогресса. Я сам, правда, не очень следую этому совету. Но попытаться все же стоит... И, конечно, удачи!

Оригинал интервью:
Ricardo Pereira Dias @ «RPDesigner»

Перевод:
Gecko

Blender: настольная книга

«Blender. Настольная книга» – это проект от журнала «FPS» по созданию полноценного русскоязычного электронного руководства по основам работы в Blender 2.6. Целевая аудитория – начинающие пользователи программы (как перешедшие со старых версий, так и начинающие знакомство с Blender «с нуля»). Книга будет представлять собой сборник статей, охватывающих различные аспекты использования Blender, скомпонованных по принципу «от простого к сложному».

Издание будет распространятся бесплатно, по лицензии Creative Commons BY SA. На данный момент активно ведется подготовка текста книги.

К работе над книгой приглашаются все желающие! На почтовый ящик редакции (gecko0307@gmail.com) принимаются статьи и уроки, а также общие советы и предложения. Кроме того, книге нужны графические материалы: авторские художественные работы, интересные скриншоты, демонстрационные рендеры, схемы, диаграммы и т.д. Весь Ваш вклад в книгу обязательно будет учтен, и Ваше имя будет указано в списке авторов.

Подробности на сайте проекта:

http://fpsmag.zymichost.com/book


LINUX
MEN ARE FROM MARS

Язык D: новости

Релиз DMD 1.074 и 2.059

Увидел свет очередной релиз компилятора DMD – 1.074 и 2.059 для D1 и D2 соответственно. В числе нововведений второй версии языка – полноценная поддержка UFCS (Uniform Function Call Syntax – единый синтаксис вызова функций). Было исправлено множество багов в самом компиляторе, рантайме и стандартной библиотеке Phobos. Некоторые устаревшие функции были помечены как не рекомендуемые к использованию. Кроме того, согласно официальному чейнджлогу, была прекращена поддержка семейства ОС MS Windows 9x.

http://dlang.org/download.html
http://www.digitalmars.com/d/2.0/changelog.html

TDPL на русском

Бестселлер Amazone Kindle, книга Андрея Александреску «The D Programming Language» («Язык программирования D») переведена на русский и уже доступна для предварительного заказа на books.ru (ISBN: 978-5-93286-205-6) по цене 739 руб.

http://www.books.ru/books/yazyk-programmirovaniya-d-827252/

Thrift поддерживает D!

Apache Thrift – это язык описания интерфейсов, который используется для определения и создания служб под разные языки программирования. Является фреймворком к удаленному вызову процедур (RPC). Сочетает в себе программный конвейер с движком генерации кода для разработки служб для таких языков как C#, C++, Cappuccino, Cocoa, Delphi, Erlang, Go, Haskell, Java, OCaml, Perl, PHP, Python, Ruby и Smalltalk. Работа над поддержкой D началась в 2011 году на Google Summer of Code, и на днях завершилась официальным включением в основную ветку проекта.

https://github.com/klickverbot/thrift
http://thrift.apache.org/

GDC переехал на GitHub

Ведется работа над новым сайтом проекта – инициатор Иэн Беклоу (Iain Buclaw) принимает идеи относительно контента сайта на свой почтовый ящик (**ibuclaw@gdcproject.org**) или на IRC-канал **#d.gdc** на FreeNode.

https://github.com/D-Programming-GDC
http://dgnu.org/
http://gdcproject.org/

На D портирован лексический и синтаксический анализатор DMD. В рамках проекта dmd-clean ведется работа по портированию фронтэнда референсного компилятора D с C++ на сам D. Другой аналогичный проект – dil – находится на стадии реализации семантического анализатора. Вполне вероятно, уже в ближайшем будущем можно ожидать появление полноценного компилятора D, написанного на D...

https://github.com/zachthemystic/ddmd-clean/
http://code.google.com/p/dil/

Появился транслятор с D на JavaScript. На данный момент он уже поддерживает многие возможности D, как то: основные типы данных, функции, циклы, ветвления, структуры, классы (включая наследование!) и обработку исключений. Сгенерированный код, правда, плохо оптимизирован – в том смысле, что использует декорацию имен D и поэтому занимает достаточно много памяти (но ведь ничто не мешает использовать автозамену).

https://github.com/adamdruppe/dtojs

CWrap – это абстракция высокого уровня для вызова функций C, призванная облегчить создание и использование привязок к C-библиотекам. Она предоставляет средства безопасной работы с памятью (строки и массивы вместо непосредственной работы с указателями, корректное высвобождение памяти во избежание утечек) а также механизм обработки исключений поверх функций C. Сгенерированные CWrap врапперы не создают мусора и не используют лишний раз динамическое выделение памяти.

https://bitbucket.org/denis_sh/cwrap

DScience портирован на D2. DScience – это международный проект по разработке свободных инструментов для исследований в области вычислительной молекулярной биологии, с учетом нынешних и будущих нужд биоинформатики.

https://gitorious.org/dscience/dscience

Goldie 0.9. Свободный набор инструментов для парсинга Goldie, совместимый с GOLD Parser Builder, обновился до версии 0.9. Была добавлена поддержка DMD 2.059 (и одновременно прекращена поддержка 2.054), внесены изменения в API и синтаксис параметров командной строки, исправлено несколько багов.

http://www.semitwist.com/goldie

Pegged – генератор парсеров. Pegged использует простую нотацию грамматик для построения полноценных парсеров – как динамически, так и во время компиляции. Процесс парсинга также может быть осуществлен во время компиляции, что лишний раз доказывает мощь метапрограммирования и CTFE в D.

https://github.com/PhilippeSigaud/Pegged

libgc-d. Появился libgc-d – биндинг к сборщику мусора Бома-Демерса-Вайзера (libgc). Эта библиотека в основном ориентирована на разработку скриптовых языков и виртуальных машин, так как в D уже предусмотрен собственный сборщик мусора. Проект находится на стадии альфа.

https://github.com/lycus/libgc-d

Mono-D 0.3.7. Mono-D – плагин для среды MonoDevelop, предоставляющий поддержку языка D в этой IDE – обновился до версии 0.3.7. Из числа нововведений стоит отметить улучшение подсветки синтаксиса, поддержку новой версии библиотек MonoDevelop 2.9.5, совместимость с DMD 2.059 и т.д.

http://mono-d.sourceforge.net
http://monodevelop.com

Visual D 0.3.32. Вышла новая версия Visual D – проекта по интеграции D в среду разработки Microsoft Visual Studio. Добавлена поддержка компилятора GDC и целевой архитектуры x64, а также нового лямбда-синтаксиса (=>) в парсер. Исправлено множество багов.

http://www.dsource.org/projects/visuald

OCILIB в Deimos. Проект Deimos обзавелся биндингом к OCILIB – открытому кроссплатформенному драйверу к реляционной СУБД Oracle Database, написанному на C. Данный биндинг предоставляет интерфейс к OCILIB 3.9.3 и распространяется по лицензии GNU LGPL.

https://github.com/D-Programming-Deimos/ocilib

Обновление DDevel.org. Социальная сеть для программистов на D, разработчиков игр и просто творческих людей ddevel.org недавно подверглась серьезному апргрейду. Был обновлен редактор сообщений, добавлена подсветка синтаксиса кода. Кнопки социальных сетей теперь отображаются на вертикальной панели сбоку.

http://ddevel.org

DPaste – онлайн-компилятор. Проект DPaste представляет собой онлайн-компилятор языка D и инструмент для совместной работы над проектами, позволяющий запускать код на D прямо в браузере! На данный момент поддерживаются DMD 2.059, DMD 1.074 и git-ревизия DMD, которая обновляется раз в три дня. Поддержка GDC планируется в ближайшем будущем.

http://dpaste.dzfl.pl

http://github.com

Язык D. Шаблоны
Часть III

Мы продолжаем разговор о обобщенном программировании на D, начатый в «FPS» №13 и 15 ('11). Шаблоны D вкупе с технологией выполнения функций во время компиляции (CTFE) – это своеобразный метаязык, позволяющий в буквальном смысле «создавать программы, которые создают программы». В первую очередь это относится к возможности «на лету» конструировать новые типы данных и передавать их между функциями.

Например, иногда требуется возвратить из функции несколько значений разных типов. Можно, конечно, заранее объявить для этого специальную структуру, но вручную это делать не всегда удобно. Тем более что возможности D позволяют это автоматизировать и привести код к симпатичному краткому виду, как мы увидим позднее.

Во многих других современных языках для таких целей используются кортежи (tuple), но D, к сожалению, пока не поддерживает возврат кортежей из функции (хотя можно сделать это через аргумент, помеченный как out, но это немного другой случай). В качестве примера приведу код на Python:

```
def getSomeValues():
    res0 = 34
    res1 = [True, False]
    res2 = 'hello'
    return (res0, res1, res2)

r = getSomeValues()
print r
print r[0]
print r[1]
print r[2]
```

В D можно сделать практически то же самое путем специализации варьируемого шаблона структуры (variadic template) списком желаемых типов. В результате мы получаем структуру с безымянными полями, доступ к которым осуществляется через оператор индексирования (например, t[0]). При этом можно использовать конструкцию «alias this», чтобы перенаправлять обращение к t. Я назвал такой шаблон Compound.

```
struct Compound(T...)
{
    T tuple;
    alias tuple this;
}
```

Теперь можно оформить нашу функцию следующим образом:

```
auto getSomeValues()
{
    Compound!(int, bool[], string) res;
```

```
    res[0] = 34;
    res[1] = [true, false];
    res[2] = "hello";
    return res;
}
```

...и вызывать:

```
auto r = getSomeValues();
writeln(r);
writeln(r[0]);
writeln(r[1]);
writeln(r[2]);
```

Можно, конечно, обойтись без шаблонов и оформить аналогичную функцию «по старинке», с использованием простой вложенной структуры:

```
auto getSomeValues()
{
    struct S
    {
        int i;
        bool[] b;
        string s;
    }

    return S(34, [true, false], "hello");
}
```

Почему, казалось бы, не делать так? Дело в том, что тут кроется важный, но малозаметный на первый взгляд просчет в дизайне программы, который рано или поздно даст о себе знать. Такие вложенные типы нельзя инстанцировать вне функции. Уолтер Брайт остроумно назвал их Волдеморт-типами (Voldemort types) – по аналогии с персонажем из «Гарри Поттера», Тем-Кого-Нельзя-Называть. Что? Вы сказали «typeof»? Если бы...

```
auto g = getSomeValues()
typeof(g) h;
```

Этот код, к сожалению, не скомпилируется. Более того, поскольку такая структура объявляется внутри контекста каждой функции, которая ее использует, заново – эти Волдеморт-типы не взаимозаменяемы, даже если их сигнатуры полностью совпадают! Например:

```
auto getSomeStruct2()
{
    struct S
    {
        int i;
        bool[] b;
        string s;
    }
    return S(22, [true, true], "foo");
}

auto a = getSomeStruct();
auto b = getSomeStruct2();
a = b;
```

Такой код не скомпилируется, так как невозможно прямое присваивание b к a (то есть, b нельзя неявно преобразовать в a). В то время как аналогичный код с использованием шаблона Compound скомпилируется без проблем – шаблон будет специализирован только один раз для всех функций, которые используют данную специализацию:

```
auto getSomeValues2()
{
    Compound!(int, bool[], string) res;
    res[0] = 22;
    res[1] = [true, true];
    res[2] = "foo";
    return res;
}

auto a = getSomeValues();
auto b = getSomeValues2();
a = b;
```

Второй важный момент: структуру Compound можно одновременно объявлять и инициализировать при помощи функции-фабрики, что на порядок упрощает ее использование и укорачивает код:

```
Compound!(T) compound(T...) (T args)
{
    return Compound!(T)(args);
}

auto s = compound(500, [true, false, false], "text");
```

Ну и, наконец, приятные побочные эффекты – куда же без них :) Такая структура является агрегатным типом, то есть ее поля можно перебирать оператором foreach.

```
foreach(i, v; s)
    writefln("%s: %s", i, v);
```

Кроме того, она взаимозаменяема с кортежем (tuple):

```
template Tuple(E...)
{
    alias E Tuple;
}

Tuple!(int, bool[], string) tu;
tu[0] = 100;
tu[1] = [true, true];
tu[2] = "hello, world!";
Struct!(int, bool[], string) st;
st = tu;
```

Фактически, это некий «гибрид» структуры и кортежа – для нее доступно свойство length, характерное для кортежей, и свойства sizeof, offsetof, характерные для обычных структур:

```
writeln(st.length);
writeln(st.sizeof);
writeln(st[0].offsetof);
writeln(st[1].offsetof);
writeln(st[2].offsetof);
```

Да будет... звук!

В одном из предыдущих номеров мы уже разобрали загрузку аудиоданных в формате WAV («Воспроизведение WAV-файлов через OpenAL», «FPS» №17 '11). При этом для воспроизведения звука использовалась популярная библиотека OpenAL, которую выбирает большинство инди-разработчиков. Но не менее интересным было бы разобраться – что представляет собой вывод звука на системном уровне? Можно ли на D написать собственный звуковой движок?

Для простоты не будем пока мыслить категориями кроссплатформенности, остановимся только на одной системе. Пусть это будет Linux. Исторически сложилось так, что за звук в Linux отвечают два разных драйвера – OSS и ALSA.

Самый старый из двух – OSS (Open Sound System) – был создан в 1992 году. В начале 90-х финский студент Ханну Савалайнен занимался программированием в области компьютерной обработки звука. Начав с написания драйверов для различных звуковых карт, Ханну задумался об объединении этих драйверов общим интерфейсом, прозрачным для всех приложений, работающих со звуком. Результатом явилось создание звуковой подсистемы для UNIX-подобной операционной системы VoxWare. Позднее ее переименовали в Unix Sound System, а потом – в Open Sound System. Поддержка OSS в ядре Linux появилась в ветке 2.4.

Однако в то время OSS выпускался под коммерческой лицензией в виде shareware – из-за закрытости кода, в ядре ветки 2.6, он был заменен на ALSA.

ALSA (Advanced Linux Sound Architecture) – современная архитектура звуковых драйверов. Включает программный микшер с очень низкой задержкой, а также поддержку множества звуковых интерфейсов – от простого стерео до профессиональных многоканальных интерфейсов.

А что же OSS? В настоящее время исходный код OSS доступен под GPL, а в недрах 4Front Technologies ведутся работы по усовершенствованию этой системы и расширению списка поддерживаемого оборудования. И, хоть ALSA является теперь стандартной звуковой подсистемой в Linux, и OSS объявлена не рекомендованной к использованию (deprecated), вопрос выбора интерфейса для воспроизведения и записи звука все еще актуален и подчас ставит в тупик не только обычных пользователей, но и программистов. Ситуация – привычная для Linux и СПО: вспомните знаменитое противостояние Qt и GTK...

Многие приложения – в том числе, большинство коммерческих игр – созданы в расчете на работу именно с OSS. Причем как по историческим причинам, так и из-за большего количества поддерживаемых OSS операционных систем.

Дело в том, что ALSA пока реализована только для Linux, в то время как OSS работает на почти на всех современных UNIX-ах – Linux, FreeBSD, OpenSolaris и т. д. Кроме того, OSS может поддерживать звуковые карты или их функции, неподдерживаемые ALSA (впрочем, обратное тоже справедливо).

На мой взгляд, есть резон использовать OSS. Хотя бы, чтобы поддерживать совместимость со старыми ядрами и другими *nix. Тем более, что ALSA предоставляет API OSS и прекрасно работает с большинством программ, написанных только для OSS. Чтобы использовать этот API, программе не нужны никакие дополнительные библиотеки – только стандартные команды UNIX. В этом плане у OSS есть свое, особенное очарование...

Основная идея OSS идея в том, что звуковое устройство – это файл (обычно /dev/dsp или /dev/audio). Любые данные, записанные в эти файлы, воспроизводятся звуковой картой. Чтение из этих файлов возвращает звуковые данные, записанные с текущего входного источника. На практике обычно используется только один из файлов устройств – как правило /dev/dsp, так как он работает с аудиоданными в широко распространенном формате PCM.

Таким образом, чтобы воспроизвести звук, мы должны записать PCM-данные в файл /dev/dsp. Не стоит пытаться делать это высокоуровневыми функциями Phobos, лучше напрямую воспользоваться системными командами UNIX. Нам понадобится модуль std.c.linux.linux, в котором объявлены эти команды.

Поскольку многие из них конфликтуют с названиями некоторых функций Phobos, этот модуль проще импортировать в уточняющее символьное обозначение:

```
import Linux = std.c.linux.linux;
```

Для работы с OSS нужно несколько специальных констант:

```
enum AFMT_U8 = 0x00000008;
enum AFMT_S8 = 0x00000040;

enum AFMT_S16_BE = 0x00000020;
enum AFMT_S16_LE = 0x00000010;

version(LittleEndian) enum AFMT_S16_NE = AFMT_S16_LE;
version(BigEndian) enum AFMT_S16_NE = AFMT_S16_BE;

enum AFMT_U16_LE = 0x00000080;
enum AFMT_U16_BE = 0x00000100;

version(BigEndian)    enum AFMT_U16_NE = AFMT_U16_BE;
version(LittleEndian) enum AFMT_U16_NE = AFMT_U16_LE;

enum AFMT_S24_LE = 0x00010000;
enum AFMT_S24_BE = 0x00020000;

version(BigEndian)    enum AFMT_S24_NE = AFMT_S24_BE;
version(LittleEndian) enum AFMT_S24_NE = AFMT_S24_LE;

enum AFMT_U24_LE = 0x00040000;
enum AFMT_U24_BE = 0x00080000;

version(BigEndian)    enum AFMT_U24_NE = AFMT_U24_BE;
version(LittleEndian) enum AFMT_U24_NE = AFMT_U24_LE;
```

```
enum AFMT_S32_LE = 0x00001000;
enum AFMT_S32_BE = 0x00002000;

version(BigEndian)    enum AFMT_S32_NE = AFMT_S32_BE;
version(LittleEndian) enum AFMT_S32_NE = AFMT_S32_LE;

enum AFMT_U32_LE = 0x00004000;
enum AFMT_U32_BE = 0x00008000;

version(BigEndian)    enum AFMT_U32_NE = AFMT_U32_BE;
version(LittleEndian) enum AFMT_U32_NE = AFMT_U32_LE;

enum SNDCTL_DSP_RESET = 0x00005000;
enum SNDCTL_DSP_SYNC = 0x00005001;
enum SNDCTL_DSP_SPEED = 0xc0045002;
enum SNDCTL_DSP_CHANNELS = 0xc0045003;
enum SNDCTL_DSP_GETBLKSIZE = 0xC0045004;
enum SNDCTL_DSP_POST = 0x00005008;
enum SNDCTL_DSP_SUBDIVIDE = 0xc0045009;
enum SNDCTL_DSP_SETFRAGMENT = 0xC004500A;
enum SNDCTL_DSP_GETFMTS = 0x8004500b;
enum SNDCTL_DSP_SETFMT = 0xc0045005;
enum SNDCTL_DSP_GETOSPACE = 0x800C500C;
enum SNDCTL_DSP_GETISPACE = 0x800C500D;
enum SNDCTL_DSP_NONBLOCK = 0x0000500E;
enum SOUND_PCM_WRITE_BITS = SNDCTL_DSP_SETFMT;
enum SOUND_PCM_WRITE_RATE = SNDCTL_DSP_SPEED;
enum SOUND_PCM_SYNC = SNDCTL_DSP_SYNC;
enum SOUND_PCM_WRITE_CHANNELS = SNDCTL_DSP_CHANNELS;
```

И еще нужна функция ioctl, которая манипулирует базовыми параметрами устройств, представленных в виде файлов:

```
extern(C)
{
    int ioctl(int d, int request, ...);
}
```

Мы рассмотрим воспроизведение WAV-файла, но я написал свой «звуковой движок» с расчетом на дальнейшее расширение – то есть, с возможностью добавления поддержки новых форматов аудио. Поэтому загрузка аудиоданных и их воспроизведение – строго разделенные друг от друга стадии. Для передачи обычных, непотоковых звуковых данных между этими стадиями, я использую структуру Sound:

```
struct Sound
{
    int channels;
    int sampleRate;
    int sampleSize; // bits per sample
    ubyte[] data;
}
```

Итак, функция для воспроизведения звука через OSS в нашем простейшем случае будет выглядеть так:

```
void ossPlay(ref Sound snd)
{
    int fd = Linux.open("/dev/dsp", Linux.O_WRONLY);
    if (fd == -1)
        throw new Exception(
            "Failed to open OSS output device");
    scope(exit) Linux.close(fd);

    int ch = snd.channels - 1;

    int fmt;
    switch (snd.sampleSize)
    {
        case 8:  fmt = AFMT_S8;     break;
        case 16: fmt = AFMT_S16_NE; break;
        case 24: fmt = AFMT_S24_NE; break;
        case 32: fmt = AFMT_S32_NE; break;
        default:
        throw new Exception(
        format("Unsupported sample size: %s bits",
            snd.sampleSize));
    }

    ioctl(fd, SOUND_PCM_WRITE_RATE, &snd.sampleRate);
    ioctl(fd, SOUND_PCM_WRITE_BITS, &snd.sampleSize);
    ioctl(fd, SOUND_PCM_WRITE_CHANNELS, &ch);

    int written = Linux.write(fd, snd.data.ptr, snd.data.length);
    if (written == -1)
        throw new Exception("write");

    ioctl(fd, SOUND_PCM_SYNC, 0);
}
```

В главной функции мы загружаем WAV-файл и вызываем нашу ossPlay:

```
int main()
{
    auto snd = wavLoad("my_sound_file.wav");
    ossPlay(snd);
    return 0;
}
```

Реализация загрузки WAV была немного изменена – ищите новую версию на сайте журнала в разделе «Файловый архив» (**http://fpsmag.zymichost.com/files.htm**). Если вы используете этот код в реалтаймовом приложении, выполняйте воспроизведение в отдельном потоке (но не передавайте дескриптор файла между потоками, если у вас не предусмотрен механизм блокировки!)

Gecko
gecko0307@gmail.com

Веб-сервер «своими руками»
Часть II

Мы продолжаем начатую в FPS №18 '12 тему о простом веб-сервере на D. В предыдущей части статьи мы рассмотрели создание сокетов и установку односторонней связи с клиентом – прием HTTP-запросов. Нам осталось написать ту часть программы, которая будет должным образом обрабатывать эти запросы и отвечать на них. Новый вариант главного цикла сервера будет выглядеть следующим образом:

```
while(true)
{
    currSock = listener.accept();
    if ((bytesRead = currSock.receive(buf)) > 0)
    {
        writeln("Client message:");
        writeln(buf[0..bytesRead]);
        string[] lines = splitLines(to!string(buf));
        if (lines.length>0)
        {
            string[] mainRequestStr = split(lines[0]);
            if (mainRequestStr[0] == "GET")
                httpGet(currSock, mainRequestStr[1]);
        }
    }
    currSock.close();
    buf.clear();
}
```

Чтобы определить тип запроса, программа разбивает полученный текст на строки, а первую строку – на слова. Первое слово первой строки – это и есть тип HTTP-запроса. Как видно, наш сервер поддерживает только один тип запроса – GET, который используется для передачи гипертекстовых (да и всех прочих) данных, расположенных на хостинге. В нашем случае это рабочий каталог сервера, но вы можете указать любой другой. Реализация ответа на GET оформлена в виде функции httpGet:

```
void httpGet(Socket s, string document)
{
    if (document == "/")
        document = "/index.html";
    if (document[0] == '/') document = document[1..$];

    writeln("[server] sending " ~ document);

    if (exists(document))
    {
        if (extension(document) == ".ico")
            httpSendDocument(s, document, "image/x-ico");
        if (extension(document) == ".jpg")
            httpSendDocument(s, document, "image/jpeg");
        else if (extension(document) == ".html")
            httpSendDocument(s, document, "text/html");
    }
```

```
    else
    {
        httpSendDocument(s, "404.html", "text/html");
    }
}
```

Семантика этой функции может меняться в зависимости от политики сервера. В данном случае мы разрешаем только передачу файлов *.html, *.ico и *.jpg.

Если запрашиваемый файл отсутствует, то передается специальный файл 404.html (с текстом ошибки 404). Непосредственно отправка файла реализована в функции httpSendDocument:

```
void httpSendDocument(Socket s,
                      string filename,
                      string mimeType,
                      int code = 200)
{
    char[] data = cast(char[])read(filename);
    char[] output =
        "HTTP/1.1 " ~
        to!string(code) ~
        " OK\r\n Content-Type: " ~
        mimeType ~
        ";\r\n\r\n" ~
        data ~
        "\r\n";
    s.sendTo(output);
}
```

Теперь можно запускать и тестировать в браузере. Под Linux для запуска сервера могут понадобиться права суперпользователя. Сервер работает в режиме бесконечного цикла, поэтому, чтобы остановить его, потребуется либо диспетчер задач (в Windows) либо команда kill с указанием PID процесса (в Linux). Чтобы узнать PID, введите команду ps:

```
$ ps -A
```

Будет выведен список процессов с их PID (Process ID) напротив. Запомните PID вашего сервера и введите команду

```
$ kill XXXX
```

где XXXX – запомненный вами номер.

Кстати, команда kill all убъет ВСЕ процессы. Она требует прав суперпользователя.

Gecko
gecko0307@gmail.com

Примечание редактора: *поскольку D2 и Phobos обновляются довольно часто, и почти всегда «ломают» обратную совместимость в именах функций, приведенный код может не скомпилироваться новейшими версиями DMD, так как статья была написана достаточно давно. Заранее приносим извинения, если вы столкнулись с подобной проблемой. Следите за изменениями и старайтесь держать код в актуальном состоянии.*

Интервью с создателем Python

Гвидо ван Россум о перспективах развития языка

Информационный IT-портал **infoworld.com** недавно опубликовал интервью с Гвидо ван Россумом – создателем и бессменным лидером одного из самых популярных языков программирования Python. Будучи сотрудником Google, ван Россум недавно выступил на конференции PyCon в Кремниевой долине, ответив на критику в адрес языка и обрисовав планы по его развитию.

В ответ на замечания о низкой производительности Python, вы всегда говорите, что ваши собственные программы работают достаточно быстро. Откуда тогда критика?

Просто люди очень часто создают что-то невероятное и даже безумное, что включает массу вычислений – например, перебор графа из миллиарда социальных контактов или анализ триллиона электронных сообщений... Неудивительно, что такие вещи, написанные на Python, выполняются не слишком быстро. Да, какие-то операции в большинстве случаев было бы эффективнее реализовать на C или C++ – но ведь необязательно переписывать на нем всю программу. Нужно понимать, что высокая производительность важна только в специфических ситуациях.

Какова сейчас ситуация с версиями Python?

На данный момент у языка две версии. Мы постепенно завершаем работу над 2.x, последняя версия из этой ветки – 2.7. Правда, версии 2.5 и 2.6 все еще достаточно широко используются. В ветке 3.x текущий стабильный релиз – 3.2. Кроме того, мы недавно выпустили альфа-версию 3.3, которая должна стабилизироваться в течение нескольких месяцев.

В чем заключается особенность Python 3?

Дело в том, что в Python 2 есть множество способов сделать одни и те же вещи по-разному. Кроме того, эта ветка имеет ряд проблем, связанных с поддержкой Юникода. Чтобы решить эти и другие вопросы, мы решили начать новую ветку – несовместимую со старыми версиями.

Если она несовместима, выходит, нужно переписывать программы?

Да, для совместимости с Python 3, в большинстве случаев, необходимо внести изменения в исходный код. Мы, кстати, предлагаем утилиту 2to3, которая делает это автоматически. Есть, правда, вещи, которые нельзя автоматизировать – их придется переписывать вручную. Это привело к тому, что сообщество выработало определенные правила написания кода на Python 2, который был бы стопроцентно совместим с Python 3.

В вашем выступлении вы рассмотрели аргументы за и против динамической типизации. Вы сказали, что, вверяя задачу поиска багов в программе компилятору, программист может существенно сократить время разработки.

Значит, вы довольны тем, что Python – динамический язык?

Абсолютно. Базовая философия языка не изменится. Я не вижу причин для отказа от динамической типизации.

Вы упоминали возможность внедрения глобального статического анализа для программ на Python. Чем это будет выгодно разработчикам?

Глобальный статический анализ позволит выявлять определенные типы багов. Кроме того, будут расширены возможности для оптимизации.

Является ли GIL (глобальная блокировка интерпретатора) препятствием к программированию многоядерных процессоров? Или это совсем другая история?

Более или менее... Если у вас несколько ядер, вы не сможете заставить каждое ядро выполнять байт-код Python параллельно с другими, так как на каждый процесс выделяется только одна GIL. В каждый момент времени в одном процессе интерпретатора Python может исполняться только один поток кода.

Оригинал интервью:
http://infoworld.com

Вы разрабатываете перспективный проект? Открыли интересный сайт? Хотите «раскрутить» свою команду или студию? Мы Вам поможем!

Спецпредложение от «FPS»!

«FPS» предлагает уникальную возможность: совершенно БЕСПЛАТНО разместить на страницах журнала рекламу Вашего проекта!! При этом от Вас требуется минимум:

- **Соответствие рекламируемого общей тематике журнала.** Это может быть игра, программное обеспечение для разработчиков, какой-либо движок или SDK, а также любой другой ресурс в рамках игростроя (включая сайты по программированию, графике, звуку и т.д.). Заявки, не отвечающие этому требованию, рассматриваться не будут.

- **Готовый баннер или рекламный лист.** Для баннеров приемлемое разрешение 800x200 (формат JPG, сжатие 100%). Для рекламных листов — 1000x700 (формат JPG, сжатие 90%). Содержание — произвольное, но не выходящее за рамки общепринятого и соответствующее грамматическим нормам. Совет: к созданию рекламного листа рекомендуем отнестись ответственно. Если не можете сами качественно оформить рекламу, найдите подходящего художника. «Голый» текст без графики и оформления не принимается.

- **Краткое описание** Вашего проекта и — обязательно — **ссылка на соответствующий сайт** (рекламу без ссылки не публикуем).

Заявки на рекламу принимаются на почтовый ящик редакции:
gecko0307@gmail.com (просьба в качестве темы указывать «Сотрудничество с FPS», а не просто «Реклама», так как письмо может отсеять спам-фильтр).

Прикрепленные материалы (рекламный лист, информация и пр.) могут быть как прикреплены к письму, так и загружены на какой-либо надежный сервер (убедительная просьба **не использовать** RapidShare, DepositFiles, Letitbit и другие подобные файлохранилища — загружайте файлы на свой сайт, блог или ftp-сервер и присылайте статические ссылки). Все материалы желательно архивировать в формате zip, rar, 7z, tar.gz, tar.bz2 или tar.lzma.

Путь Программиста

Обретение дзен...

Мудрость бывает краткой. Выражая свои мысли, люди могут произносить тысячи слов, но в лучшем случае только одно из них наделено истинным смыслом. А остальные зачастую только мешают увидеть этот смысл. Так же и в программировании – тонны лишнего, избыточного кода могут быть заменены одной-двумя строками, в которых будет заключен весь необходимый смысл. Если, конечно, вы сумеете найти эти строки...

В данном тексте я не приведу ни одной строки кода. Может даже возникнуть вопрос: а о программировании ли тут говорится? И да, и нет. Искусство программирования – это не наука, в отличие от самого процесса программирования. Это область интуиции и просветления, а не законов и формул...

Учебники и лекции помогут изучить и понять алгоритмы, структуры данных, принципы работы вычислительной машины, особенности разных языков и операционных систем. Но никто вам не объяснит, как писать действительно полезные, надежные и удобные программы. Это вопрос внутренней дисциплины и способности к созерцанию, способности уловить момент... Поэтому описанные ниже идеи применимы вообще чуть ли не к любой области человеческой деятельности.

Всем известнен принцип KISS:

Keep It Short and Simple

Или, как гласит дзен Python, «простое лучше, чем сложное». Задумывались ли вы над этими словами? Проектируя различные системы, я часто говорю себе «Стоп! Получается сложнее, чем должно быть!» и возвращаюсь к началу, чтобы понять, что пошло не так.

Это величайшее искусство: найти ту «точку равновесия», когда дальнейшее усложнение приведет только к запутыванию кода и не даст проекту (или модулю проекта) ничего положительного. Экзюпери сказал по этому поводу так: «Совершенство достигается не тогда, когда уже нечего прибавить, но когда уже ничего нельзя отнять...»

В этой связи интересен принцип YAGNI, который несколько перефразирует KISS:

You Ain't Gonna Need It

«Оно тебе не понадобится». Помни об этом! Добавляй новое, только если в этом возникает необходимость – а не просто «чтобы было»...

Мне лично запомнились слова Линуса Торвальдса: *«Никто не должен браться за большой проект. Начинайте с маленького тривиального проекта, и никогда не ждите, что он станет большим. Если вы будете этого ждать, вы переусложните проект и будете считать его более значимым, чем он есть на этом уровне. Или, хуже, вас может отпугнуть объем работы, которую вы себе вообразили. Поэтому начинайте с малого и думайте о деталях. Не думайте о картине в целом и крутой конструкции. Если проект не решает текущую потребность, он почти наверняка переусложнен...»*

KISS и YAGNI могут быть сведены к принципу Ричарда П. Гэбриела:

Worse is Better

«Чем хуже, тем лучше». Парадоксально, не правда ли? Как правило, при развитии какого-либо качества, приходится жертвовать другими. Этого не избежать. Становясь лучше, мы становимся... хуже. Поэтому были разработаны четыре простых рекомандации – с каким приоритетом нужно жертвовать:

1. *Простота*: реализация и интерфейс должны быть простыми. Простота реализации даже несколько важнее простоты интерфейса. Простота – самое важное требование при выборе дизайна.

2. *Правильность*: дизайн должен быть правильным во всех видимых проявлениях. Простой дизайн немного лучше, чем правильный.

3. *Логичность* (последовательность): дизайн не должен быть слишком нелогичным. Иногда можно пожертвовать логичностью ради простоты, но лучше отказаться от тех частей дизайна, которые полезны лишь в редких случаях, чем усложнить реализацию или пожертвовать логичностью.

4. *Полнота*: дизайн должен охватывать как можно больше важных ситуаций. Полнотой можно жертвовать в пользу остальных качеств и обязательно нужно жертвовать, если она мешает простоте. Логичностью можно жертвовать в пользу полноты, если сохраняется простота.

Дух UNIX (или UNIX-way), о котором много говорят, но никто до конца не понимает – базируется именно на этой идее. На идее простоты. Если вы усложняете – то хотя бы делайте это оправданно. И, что особо важно, – без «сюрпризов» для пользователя. Поэтому, например, просмотровщик изображений, который умеет воспроизводить видео – не UNIX-way. Программа просмотра видео, которая умеет записывать DVD – не UNIX-way. Программа для записи DVD, которая включает инструменты видеомонтажа – не UNIX-way. И так далее...

Философию UNIX, на мой взгляд, лучше всего сформулировал Эрик С. Рэймонд в своих «Правилах»:

- **Правило модульности**: Пишите простые части, соединяемые понятными интерфейсами.
- **Правило ясности**: Ясность лучше заумности.
- **Правило композиции**: Разрабатывайте программы так, чтобы их можно было соединить с другими программами.
- **Правило разделения**: Отделяйте интерфейс от движка.

- **Правило простоты**: Нацельтесь на простоту; добавляйте сложность, только где необходимо.
- **Правило экономности**: Пишите большую программу только когда другими средствами выполнить необходимую задачу не удастся.
- **Правило прозрачности**: Разрабатывайте прозрачные программы для облегчения последующего пересмотра и отладки.
- **Правило надежности**: Надежность – дитя прозрачности и простоты.
- **Правило представления**: Храните знания в данных так, чтобы логика программы была тупой и надежной.
- **Правило наименьшего удивления**: При разработке интерфейса всегда делайте как можно меньше неожиданных вещей.
- **Правило тишины**: Если программе нечего сказать, пусть лучше молчит.
- **Правило восстановления**: Если надо выйти из строя, делайте это шумно и как можно быстрее.
- **Правило экономии**: Время программиста дорого; сократите его, используя машинное время.
- **Правило генерации**: Избегайте ручного набора кода; при любом удобном случае пишите программы, которые бы писали программы.
- **Правило оптимизации**: Сначала – опытный образец, и только потом – «причесывание». Добейтесь стабильной работы, потом оптимизируйте.
- **Правило многообразия**: Отвергайте все утверждения об «единственно правильном пути».
- **Правило расширяемости**: Разрабатывайте для будущего. Оно наступит быстрее, чем вы думаете.

Если же вам нужна философия UNIX, выраженная одним предложением, то это, несомненно, изречение Дуга Мак-Илроя:

Пишите программы, которые делают что-то одно и делают это хорошо.

«Однажды ученик, отлаживая программу, грубо нарушил медитацию мастера. Он удивленно вскричал:
– Моя программа не падает!
Мастер, не задумавшись ни на секунду, сказал:
– Значит, в ней ошибка.
– Но как я могу поймать ошибку, если программа не падает?
– Если программа падает, ошибку поймает любой. Истинное Дао в необнаруживающих себя ошибках...»

Gecko
gecko0307@gmail.com

Linux: полезные команды

У всякого пользователя Linux со временем скапливается множество типовых консольных команд, которые он использует чаще всего. Ваш покорный слуга – не исключение. Хочу привести здесь некоторые из своей «коллекции». Не факт, правда, что они вам пригодятся – но, как говорится, дареному коню в зубы не смотрят...

Смонтировать сетевой каталог Windows (в окне запуска – Alt+F2):

```
smb://xxx.yyy.zzz.www/folder_name
```

где xxx.yyy.zzz.www – IP-адрес узла.

«Вычленить» звуковую дорожку из видео при помощи MPlayer:

```
$ mplayer -vc null -vo null -ao pcm -benchmark video.avi
```

Раскодировать сжатый звук в WAV при помощи MPlayer:

```
$ mplayer -ao pcm music.mp3
```

Сконвертировать все изображения JPG в текущем каталоге в формат PNG, используя ImageMagick:

```
$ mogrify -format jpg *.png
```

Разбить большой видеофайл в формате MKV на части по 700 Мб:

```
$ mkvmerge --split size:700m -o myfile.mkv
```

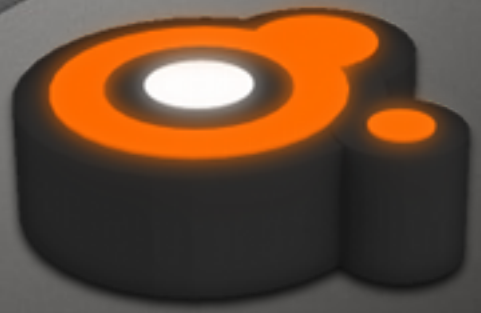
Найти текст рекурсивно во всех файлах указанного расширения в текущем каталоге:

```
$ grep -i -r "SomeText" ./*.d
```

Gecko
gecko0307@gmail.com

Это все!

Надеемся, номер вышел интересным. Если Вам нравится наш журнал, и Вы хотели бы его поддержать – участвуйте в его создании! Отправляйте статьи, обзоры, интервью и прочее на любые темы, касающиеся игр, графики, звука, программирования и т.д. на gecko0307@gmail.com.



http://fpsmag.zymichost.com