ЗДРАВСТВУЙТЕ, меня зовут Хабаров Александр, **ТЕМА** моей работы - **«КОНТЕКСТНО-ОРИЕНТИРОВАННАЯ СИСТЕМА РЕНДЕРИНГА ДВУХМЕРНОЙ ГРАФИКИ ДЛЯ OS X»**.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

В современных приложениях OS X для рендеринга двухмерной графики используется разработанный **APPLE** фреймворк **CORE GRAPHICS**. Основная **ПРОБЛЕМА** использования этого фреймворка заключается в его **НИЗКОУРОВНЕВОСТИ**. Несмотря на **ПОЛУЧЕНИЕ ПОЛНОГО ДОСТУПА** к графике системы, **ОБЪЕМ КОДА** оказывается весьма большим, отсутствует **ООП И СОБЫТИЙНОСТЬ**, а **СЛОЖНОСТЬ РАЗРАБОТКИ** программ сильно возрастает. Перед нами встала **НЕОБХОДИМОСТЬ** создания **ОБОЛОЧКИ** над Core Graphics. АНИМАЦИЯ, ПОЖАЛУЙСТА. **WRAPPER** предоставляет возможность использовать **ООП** с умеренным **ОГРАНИЧЕНИЕМ ДОСТУПА** к системе. В то же время, **НЕ ИМЕЕТ СМЫСЛА** делать весь код **ОБЪЕКТНО-ОРИЕНТИРОВАННЫМ**: иногда достаточно **C-API** для придания необходимого удобства системе.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

Главной **ЗАДАЧЕЙ** на протяжении всего проекта оставалось осуществление **ОТОБРАЖЕНИЯ** графических **ПРИМИТИВОВ**. Но для полной функциональности необходимо было предоставить пользователю **ДОСТУП** к **КАЖДОМУ ПИКСЕЛЮ** контекста. Core Graphics **ПОТОКОБЕЗОПАСЕН**, однако доступ к **МАССИВУ ПИКСЕЛЕЙ КОНТЕКСТА** и изменение **ПЕРЕМЕННЫХ-СТАТУСОВ** должны быть **ЗАЩИЩЕНЫ** отдельно. Поэтому одной из наших главных **ЗАДАЧ** являлась разработка механизмов **СИНХРОНИЗАЦИИ** потоков приложения.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

На этом слайде рассмотрим пример **СОЗДАНИЯ КОНТЕКСТА БИТОВОЙ КАРТЫ** из **МАССИВА** пикселей (self.data), **ИСПОЛЬЗУЯ** низкоуровневый API **CORE GRAPHICS**. Также мы передаем **РАЗМЕРЫ ОБЪЕКТА**, информацию о кол-ве **БИТ НА ТОЧКУ ИЗОБРАЖЕНИЯ**, способы **ПЕРЕДАЧИ ЦВЕТА** и **ХРАНЕНИЯ ДАННЫХ**. Рендеринг пикселей осуществляется исключительно **ЧЕРЕЗ МАССИВ**, поскольку изменение отдельных пикселей с помощью методов контекста сильно снижает **ПРОИЗВОДИТЕЛЬНОСТЬ** программы. Этот код

весьма **СЛОЖЕН**, но с помощью **WRAPPER'A** пользователю не нужно задумываться об его устройстве.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

При каждом **СОБЫТИИ ПЕРЕРИСОВКИ** мы **ВЫНУЖДЕНЫ** создавать объект **ИЗОБРАЖЕНИЯ CORE GRAPHICS** из содержимого нашего контекста. Этот объект необходимо **ОСВОБОЖДАТЬ** вручную, несмотря на использованиие его при инициализации NSImage. Мы пользуемся **ВЫСОКОУРОВНЕВЫМИ** методами **NSIMAGE** для непосредственного **ОТОБРАЖЕНИЯ** содержимого контекста на экране. При выполнении этой операции наше изображение **НАКЛАДЫВАЕТСЯ** на содержимое выскокуровневого контекста графики, передаваемого нам при таком событии.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

На таблице вы можете видеть **УРОВНИ** фреймворков операционной системы OS X. Чем **ВЫШЕ** находится фреймворк, тем более **ВЫСОКОУРОВНЕВЫЕ** технологии он использует. **CORE GRAPHICS** является частью низкоуровневого **МЕДИА-РАЗДЕЛА** системы и полностью **НАПИСАН НА С**.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

На следующих слайдах приведены **СРАВНЕНИЯ** способов рендеринга некоторых геометрических **ФИГУР** через **CORE GRAPHICS** и **OBJECTIVE-C WRAPPER**. Мы видим сильное **УЛУЧШЕНИЕ** кода от C к Objective-C. При рендеринге через Objective-C класс более не нужно тратить время на заполнение отдельных **СТРУКТУР** - у каждой функции созданы различные **ПЕРЕГРУЗКИ**. Соответственно, объем кода сильно уменьшается. В данном примере вместо написания 3 функций нужно написать всего лишь 1. Такие изменения происходят **НЕ ТОЛЬКО** при отображении **ПРЯМОУГОЛЬНИКА**.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

На этом слайде мы видим реализацию рендеринга **ЛОМАНОЙ**. Рендеринг всех геометрических фигур сильно **УЛУЧШЕН** с помощью парадигм **ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ**.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

Для **РАБОТЫ С ЦВЕТОМ** создается отдельный **КЛАСС**, взаимодействующий с wrapper'ом. Массив

пикселей и класс цвета поддерживают **ПРОЗРАЧНОСТЬ**, что позволяет добиваться множества сложных эффектов простыми способами.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

API WRAPPER'A более понятен и **ПРОСТ** в использовании, нежели Core Graphics. Это видно на примере переработанной системы **УСТАНОВКИ ПАРАМЕТРОВ** рендеринга. На данном слайде приведена реализация работы с **ЦВЕТОМ ТЕКСТА И ТОЛЩИНОЙ ЛИНИЙ**.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

При создании **ООП wrapper'a** необходимо было сохранить самые **важные возможности Core Graphics**, включая: рендеринг **ГЕОМЕТРИИ**, создание дополнительных **КОНТЕКСТОВ**, работу с **ИЗОБРАЖЕНИЯМИ**, задание **ПАРАМЕТРОВ** рендеринга. Также мы добавили некоторые новые возможности для управления **СОСТОЯНИЕМ ХОЛСТА**.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

В отдельных случаях для увеличения **ПРОСТОТЫ** API используются **С-ФУНКЦИИ**. Один и тот же код

оказывается совершенно разным по уровню сложности при использовании Core Graphics & TX Mac. Таким образом создается **КОНТЕКСТ**. В случае **CORE GRAPHICS** необходимо задать множество **НИЗКОУРОВНЕВЫХ ПАРАМЕТРОВ**, многие из которых требуют отдельного создания. **TX MAC** придает удобства работе с графикой системы, устанавливая большинство параметров **АВТОМАТИЧЕСКИ**.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

Рендеринг **ИЗОБРАЖЕНИЙ** в TX Mac также сильно **УЛУЧШЕН**. Для рендеринга **СЛОЖНЫХ ОБЪЕКТОВ** в Core Graphics необходимо иметь представление о работе системы с **МАТРИЦАМИ**. Иногда приходится менять **СИСТЕМЫ КООРДИНАТ** и переворачивать объекты, что реализовано **АВТОМАТИЧЕСКИ** за счет C-wrapper'a. Уменьшение объема кода также весьма заметно.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

С-ФУНКЦИИ используются в качестве **ОБОЛОЧКИ** не только над **С-КОДОМ**, но и над **OBJECTIVE-C-КОДОМ**. Так изменено задание **ЦВЕТА ЛИНИЙ**. Несколько перегрузок функции позволяет задавать цвет

различными способами, что заметно увеличивает удобство использования системы.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

Все оболочки соединены в едином **ФРЕЙМВОРКЕ**. Предусмотрены **ШАБЛОНЫ ПРОЕКТОВ**, помогающие заметно ускорить работу на начальном этапе.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

Пример программы, выводящей текст **“HELLO, WORLD!”**. Несмотря на **ПРОСТОТУ** написания программы, в нее **АВТОМАТИЧЕСКИ** интегрированы **FACEBOOK**, **TWITTER** и **СЕРВИСЫ APPLE**.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

На данном слайде рассмотрим более **ПРОДВИНУТЫЙ** пример программы. **TXHELL** позволяет нам применять различные **ФИЛЬТРЫ** к изображению в реальном времени. В данном случае это **BUMP DISTORTION**. Каждый раз перед рендерингом мы пользуемся функцией **DISTORTIONS** для применения этого эффекта к изображению.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

Итак, были реализованы **С И OBJECTIVE-C WRAPPER'Ы** над **CORE GRAPHICS**, а также высокоуровневый **ФРЕЙМВОРК** для **РАЗРАБОТКИ** под OS X, включающий в себя также **API**: работы с **КОНСОЛЬЮ**, распознавания **ВВОДА** с **УСТРОЙСТВ** и работы с **ОПЕРАЦИОННОЙ СИСТЕМОЙ**.

ДЕМОНСТРАЦИЯ

Откроем **XCODE** и выберем **ШАБЛОН ПРОЕКТА** TX Mac. **СОЗДАДИМ** проект. Откроем **MainMenu.xib** и изменим **размеры окна** через **редактор интерфейсса**. Нам больше не нужно использваоть функцию **txCreateWindow**. Откроем **txmain.h**. Создадим **ГЛАВНЫЙ ЦИКЛ** программу и **ПЕРЕМЕННУЮ СОСТОЯНИЯ** мыши. Установим **ПАРАМЕТРЫ РЕНДЕРИНГА**. Поскольку TX Mac пока не поддерживает все функции контекста Core Graphics, их можно использовать напрямую с помощью функции **TXDC**. Таким образом мы установили **ЗЕЛЕНЫЙ ЦВЕТ** фигур и **ТЕНЬ**. Если нажата **ЛЕВАЯ КНОПКА** мыши произведем **РЕНДЕРИНГ КРУГА** в зависимости от статуса кнопки. Мы **ОБНОВЛЯЕМ СТАТУС** в зависимости от возвращаемого значения функции **TXMOUSEBUTTONS**. **СКОМПИЛИРУЕМ** проект. Мы видим **РЕНДЕРИНГ** полупрозрачных кругов различных цветов с тенями при нажатии мыши.

СЛЕДУЮЩИЙ СЛАЙД, ПОЖАЛУЙСТА

СПАСИБО ЗА ВНИМАНИЕ. Сейчас я готов **ОТВЕТИТЬ** на все ваши **ВОПРОСЫ**.