mppg
C++ библиотека для создания интерфейса при численном моделировании физических процессов

Прууэл Эдуард Рейнович, pru@hydro.nsc.ru

10 ноября 2006 г.

Содержание

Это появилось, это не могло не появиться

1 Введение

mppg (Modeling Physical Phenomena Graphical User Interface) — мультиплатформенная графическая C++ библиотека для создания пользовательского интерфейса при численном моделировании физических процессов (окно для рисования примитивов, редактируемые поля ввода/вывода, ...). Библиотека проста в освоении и использовании, достаточно функциональна, позволяет сосредоточиться на численных вычислениях и не отвлекаться на разработку интерфейса. Является хорошим решением для школьников, студентов и научных работников, для всех кто занимается отладкой расчетной программы или тестированием численного метода и кому необходимо следить и управлять проводимым расчетом.

На 11.11.06 доступна версия 2.9.10. Ее можно найти в сети интернет: `ftp://ancient.hydro.nsc.ru:/local/home_page/mppg/mppg_ru.htm`.

mppg основана на FLTK toolkit и для работы требует установленной FLTK.

Библиотека не предоставляет никаких средств для самих численных методов, это исключительно средство для создания интерфейса.

1.1 Возможности

- Унифицированный интерфейс: окно для рисования графиков, таблица с редактируемыми полями ввода вывода.
- Легкое добавление ввода и вывода значений переменных программы.
- Набор примитивов для рисования на экране с физическими масштабами.
- Сохранение изображения в файл (eps, tiff, png).

1.2 Поддерживаемые платформы

- X (UNIX) gcc
- Microsoft Windows: Cygwin, MinGW, Microsoft Visual C++.

1.3 Лицензия

Исходный код mppg библиотеки полностью свободен, любой человек в праве использовать его произвольным образом в открытых и закрытых проектах. При распространении приложений, статически слинкованных с FLTK, а это происходит по умолчанию, ознакомьтесь с лицензионным соглашением по использованию FLTK. Ограничения FLTK лицензии незначительны, и в первом приближении никак вас не ограничивают.

2 Обзор

2.1 Компилирование и установка библиотеки

Для сборки mppg библиотеки необходима FLTK, к сожалению, в современности, требуется установка неофициальной ветки FLTK-1.2. Исходные файлы этой библиотеки можно най-

ти на оригинальном сайте www.fltk.org или ftp://ancient.hydro.nsc.ru/local/home_page/fltk. Рекомендуется пользоваться последним вариантом.

2.1.1 GNU инструментарий

Все традиционно:

```
cd ./mppg-2.9.6
./configure
make
make install
```

После успешного выполнения в директории lib появится файл libmpp.a, а в директории tests будут собраны тесты.

Для облегчения компилирования собственной программы рекомендуется использовать гну утилиты. В директории user лежит заготовка проекта (makefile, mppg_hello.cpp) для создания простого mppg приложения. При обновлении используемой версии mppg рекомендуется и обновить makefile в своем проекте.

2.1.2 Microsoft Visual C++

Запустите проект visualc/mpp_lib.dsw, соберите библиотеку и тесты, самостоятельно скопируйте заголовочные файлы и библиотеку в директории компилятора.

2.2 Именование

Все открытые части mppg находятся в пространстве имен mpp.

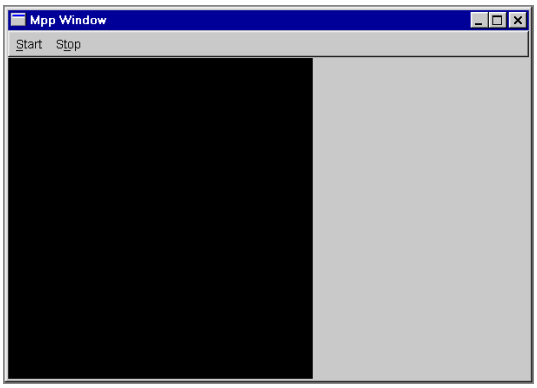
- Имена классов начинаются с заглавной буквы: Table, Mpp_window.
- Имена функций пишутся сточными буквами, части составного названия разделяются через подчерк: add_item(...);
- Все заголовочные файлы находятся в каталоге <mpp/...>.

2.3 Пример программы

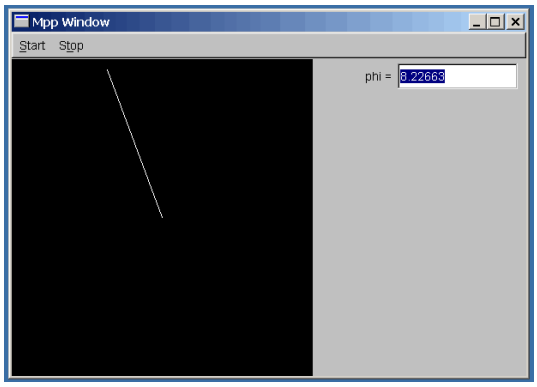
Приведем пример простейшего приложения, использующего mppg.

```
// Simple mpp application
#include <mpp/mpp_window.h>
using namespace mpp;
int main(){
   Mpp_window win; // конструируется окно
   return run();   // запускается обработчик событий
}
```

Его внешний вид изображен на рис. 1, а.



а



б

Рис. 1: Примеры внешнего вида mppg приложения.

2.4 Взаимодействие частей mppg приложения

Приведем пример более полноценного приложения.

```
#include <cmath>
#include <mpp/mpp_window.h>
#include <mpp/draw.h>

using namespace mpp;

Scale scale(-1, 1, -1, 1);
double phi=0, dphi=1e-6;

void ud(){                          // что будем рисовать
    set_scale(scale);               // устанавливаем масштаб
    set_color(FL_WHITE);            // устанавливаем цвет для рисования
    line(0,0,cos(phi),sin(phi));    // рисуем линию
}

void f(){                           // расчетная функция
    while (may_i_repeat())){        // пока позволяют продолжать
         phi+=dphi;                 // выполняем расчет
         ask_update();              // просим при случае перерисовать экран
    }                               // и обновить таблицу
}

int main(){
    Mpp_window win;    // создаем основное окно с таблицей и областью для рисования
    win.canvas.add_shape(new User_draw(ud)); // добавляем функцию для рисования
    win.table.add_item("phi = ", phi);       // поле для редактирования переменной
    set_work(f);                    // запускать f в отдельном потоке
    start();                        // запускаем расчетную функцию
    return run();                   // запускаем обработчик событий
}
```

Mppg приложение может состоять из неограниченное количество окон (объектов типа `Mpp_window`, `Table` и `Canvas`).

У mppg приложения возможны только два состояния: "расчет" и "редактирование полей ввода/вывода". Никаких других состояний быть не может. Переключение осуществляется посредством кнопок "Start" и "Stop" или из программы вызовом функциями `start()` и `stop()`.

В состоянии "расчет" запускается расчетная функция, в которой и происходят основные вычислительные действия приложения. По таймеру обновляются все объекты (таблица, канвас ...), помеченные как требующие обновления. Какой либо из объектов можно пометить, как требующий обновления вызвав его метод `ask_update()`. Можно пометить все объекты в программе вызвав функцию `ask_update()`.

При перерисовке canvas вызывает метод redraw(), для каждой из фигур (Shape) в своем списке. При этом, каждая из фигур перерисовывается в соответстви с новыми данными. В mppg есть набор часто используемых графических объектов. Если этот набор не удовлетворяет, можно использовать `User_draw`. Этот класс позволяет рисовать непосредственно с помощью графических примитивов mppg и FLTK.

2.5 Особенности и альтернативы

Поддерживается два способа помещения расчетной функции в приложение. Расчетная функция находится в отдельном потоке исполнения (`set_work(f)`), расчетная функция находится в одном потоке с интерфейсом (`set_work2(f)`), периодически вызывается, исполняя небольшую часть расчета. Каждый из способов имеет свои особенности, и в зависимости от конкретных обстоятельств, можно воспользоваться тем или другим, или обоими.

Если расчетная функция выполняется в основном потоке, это снимает проблему синхронизации, но накладывает ограничение на время ее выполнения; пока она выполняется приложение не реагирует на сообщения. Предполагается, что все используемые расчетной функцией ресурсы имеют глобальную область видимости. Это не самое удобное решение, его преимущество — простота взаимодействия расчетной функции с приложением. Расчетная функция может вызывать любые функции, в том числе `Table::ask_update()`, `Canvas::ask_update()`, `ask_update()`.

Более удобным является случай, когда расчетная функция находится в отдельном потоке. Часть расчетного кода обвернута в цикл повторяющийся пока можно (`while(may_i_repeat(){...}`). После его завершения необходимо оставить все используемые глобальные данные в валидном состоянии и освободить все занимаемые ресурсы. *Вся ответственность за синхронизацию лежит на пользователе библиотеки.* `may_i_repeat()` вернет `false` после нажатия кнопки "stop".

3 Классы и функции mppg

3.1 Классы

Иерархия классов

- Widget
  - Group
    - Window
      - Canvas (Fl_Window)
      - Table (Fl_Window)
      - Mpp_Window (Fl_Window)
  - Input (Fl_Input)
- Manager (Fl_Window)

- Scale
- Shape
  - Bitmap
  - Circle
  - Grid
  - Hist
  - Orbit
  - Plot
  - Plot_a
  - Plot_v
  - Plot_l
  - Roll
  - User_draw

3.1.1 Базовые классы

Widget #include <mpp/mpp.h>
Базовый класс для всех виджетов mppg. При конструировании всех унаследованных классов вызывается его конструктор, в котором происходит регистрация, это позволяет централизованно управлять объектами.

Widget()
virtual ~Widget()
virtual void ask_update()
virtual void update()
virtual void set_value()

virtual void ask_update()
Метит виджет, как требующий обновления.
virtual void update()
Обновляет виджет в соответствии с текущими значениями данных в программе.
virtual void set_value()
Считывает значение из поля ввода и обновляет состояние соответствующей переменной в программе.

Group #include <mpp/mpp.h>
Группа виджетов, которыми можно манипулировать как одним целым. При вызове методов ask_update(), update() и set_value() рекурсивно действие переносится на всех членов группы.

Group()
void add(Widget *)
void mpp_begin()
void mpp_end()

Window #include <mpp/mpp.h>
Предок всех окошек mppg, при его конструировании указатель на него отдается менеджеру приложения, что позволяет в дальнейшем манипулировать ими через него.

3.1.2 Основные графические виджеты

Canvas #include <mpp/canvas.h>
Виджет обеспечивающий пространство для рисования.

Canvas	add_shape
~Canvas	redraw

Canvas(const char * name)
Canvas(int w=500, int h=500, const char * name="Canvas")
Canvas(int x, int y, int w=500, int h=500, const char * name="Canvas")
Конструируется Canvas в виде отдельного окошка с именем *name*, именно через это имя происходит идентификация окна в менеджере; конструктор по умолчанию; конструируется Canvas в виде виджета в текущем окне.
~Canvas()
void add_shape(Shape *sh)
void add_shape(Shape &sh)
Добавляет форму в список отображаемых объектов. Ничего не перерисовывает. Это единственный метод, который влияет на то что будет рисоваться.

Table #include <MPP/table.h>
Меню с редактируемыми полями ввода (таблица).

Table	update
add_item	set

Table(const char * name)
Table(int w=250, int h=300, const char * name="Table")
Table(int x, int y, int w, int h, const char * name=0)
Конструирует пустую таблицу.
template<class T> void add_item(const char * name, T* var, int deact=0)
template<class T> void add_item(const char * name, T& var, int deact=0)
Добавляет в таблицу редактируемую строку со значением переменной по адресу *var*. Флаг *deact* указывает на создание нередактируемого поля. Шаблон инстанцируется для всех типов имеющих операторы чтения и записи в поток.
void update()
Обновляет строки в таблице, считывая значения из переменных.
void set_value()
Обновляет глобальные переменные, считывая их значения из таблицы.

Mpp_window #include <mpp/mpp_window.h>
Основное окошко приложения. Содержит область для рисования и таблицу.

Mpp_window	Canvas canvas
~Mpp_window	Table table

Mpp_window(int w=600, int h=400, const char *name="Mpp Window")
Mpp_window(const char *name)
Mpp_window::~Mpp_window()

Manager #include <MPP/manager.h>
Окно для управления приложением. Позволяет отображать другие окошки, запускать и останавливать счет. Автоматически создается при выполнении функции mpp::run().
Manager(int w=300, int h=200, const char * name="MPP manager")

3.1.3 Графические объекты

Библиотека предоставляет набор часто используемых фигур для отображения расчетных данных. Все они являются потомками абстрактного класса Shape, при перерисовке канвас пробегает по списку фигур и вызывает у каждой метод draw, который и производит необходимое рисование. Новый графические объекта добавляются в список вызовом одного из методов канваса: add_shape(Shape *), add_shape(Shape &).

Shape #include <MPP/draw.h> Абстрактный класс для рисования относительно сложных графических объектов (*hist*, *orbit*, ...). Предполагается использовать их посредством *canvas.add_shape*.

Bitmap #include <MPP/draw.h> Рисуется двумерный массив оттенками серого.
Bitmap(const double *, int w, int h, double C1, double C2, const Scale *)

Circle #include <MPP/draw.h> Рисуется окружность с заданными параметрами.
Circle(Float x, Float y, Float R, Fl_Color, const Scale *)

Grid #include <MPP/draw.h> Рисуется прямоугольная сетка от xmin с шагом dx пока внутри области ограниченной масштабом. Аналогично по y.
Grid(double xmin, double dx, double ymin, double dy, const Scale *)

Hist #include <MPP/draw.h> Гистограмма.
Hist(unsigned int nc, double xmin, double xmax, Fl_Color color1, const Scale *, int legs=0)
Строит гистограмму из *nc* каналов на промежутке от *xmin* до *xmax*.
void set(const double *x, unsigned int sz)
Распределяет значения из массива *x* длины *sz* по каналам гистограммы.
void add(const double *x, unsigned int sz)
Аналогично *set*, но добавляет новые события к старым значениям в каналах.

Orbit #include <MPP/draw.h> Рисует траекторию точками.
Orbit(int sz_all, int sz_head, Fl_Color color_head, Fl_Color color_tail, const Scale*)
Рисует траекторию из sz_all точек, первые sz_head цветом color_head, остальные цветом color_tail.
void add(double x, double y)
Добавляет новую точку к траектории, если количество точек становится больше sz_all, отбрасывает старые.
void reset()
Очищает траекторию.

Plot #include <MPP/draw.h> Рисует один итерратор относительно другого.
template <class T> class Plot(T x1, T x2, T y1, Fl_Color color, const Scale *)

Plot_a #include <MPP/draw.h> Рисует один массив относительно другого.
Plot_a(const double *x, const double *y, int sz, Fl_Color color, const Scale *)

Plot_v #include <MPP/draw.h> Рисует один вектор относительно другого.
Plot_v(const std::vector<double> &x, const std::vector<double> &y, Fl_Color color, const Scale)

Plot_l #include <MPP/draw.h> Рисует список пар точек.
Plot_l(const std::list<std::pair<mpp::Float, mpp::Float> > & pointl, Fl_Color color, const Scale &)

Roll #include <MPP/draw.h> Множество точек сдвигающееся влево.
Roll(Fl_Color color1, const Scale *)
void add(double x, double y)
Добавляется новая точка.
void reset()
Очищает массив точек.

User_draw #include <MPP/draw.h> Пользователь сам, с помощью графических примитивов, определяет в функции *ff* что будет рисоваться. В этой функции следует не забывать устанавливать необходимые атрибуты рисования (set_scale(scale), set_color(color), ...).
User_draw(void (*ff)())

3.1.4 Stuff

Scale #include <MPP/scale.h>
Класс позволяющий рисовать в canvas"e" в вещественных координатах. Масштаб не принадлежит никакому окну, но адаптирует работу графических примитивов для любого окна с помощью функций void set_scale(const Scale *), void set_scale(const Scale &).

Для преобразования координат и промежутков из физических единиц в оконные используются функции: inline int transx(double x), inline int transy(double y), inline int scalex(double x), inline int scaley(double y).
Scale(Float x1, Float x2, Float y1, Float y2, bool xy=false)
Конструирует масштаб с соответствующими вещественными координатами в окне. Аргумент xy устанавливает одинаковое разрешение по горизонтали и вертикали.
void set(Float x1, Float x2, Float y1, Float y2, bool xy=false)
Устанавливает новые параметры для масштаба.

3.2 Функции

file_diaolog	scalex	set_work	transx
grabbing	scaley	start	transy
plot	set_scale	stop	
run	set_update_time	set_work	

3.2.1 Графические примитивы

Набор графических примитивов для рисования можно использовать только в User_draw. Прежде чем использовать примитивы, следует выставить атрибуты рисования: set_color(Fl_Color), set_scale(const Mpp_Scale *). Без последнего масштабы могут не работать.

plot #include <MPP/draw.h>
void point(Float x, Float y)
void point(Float x, Float y, Fl_Color color)
void pointb(Float x, Float y)
void pointb(Float x, Float y, Fl_Color color)
void line(Float x, Float y, Float x1, Float y1)
void line(Float x, Float y, Float x1, Float y1, Fl_Color color)
void circle(Float x, Float y, Float r)
void circle(Float x, Float y, Float r, Fl_Color color)

Функция point ставит точку размером 1+2 point_size, pointb ставит точку размером 1+2 point_size_b.

extern int point_size=0
extern int point_size_b=1

Размеры точек по умолчанию. При необходимости их можно изменить.

template <class T> void plot(T x1, T x2, T y1)

x1, x2 — итераторы на начало и конец контейнера с x координатами, y1 — итератор на начало контейнера со значениям y.

template <class T> void plot(const T& f, double xl, double xr, int n)

Рисуется объект функция (f(double)) на промежутке (xl, xr) по n точкам.

void plot(double(*)(double), double xl, double xr, int n)

Рисуется функция (f(double)) на промежутке (xl, xr) по n точкам.

3.2.2 Работа с масштабами

Набор функций для преобразования координат и промежутков из физических единиц в экранные. Правильно работают только после установления атрибута set_scale.

scalex #include <MPP/scale.h>
int scalex(Float x)
int scaley(Float y)

transx #include <MPP/scale.h>
int transx(Float x)
int transy(Float y)

set_scale #include <MPP/scale.h>
void set_scale(const Scale *)
void set_scale(const Scale &s)

3.2.3 Управление проектом

set_work #include <MPP/mpp.h>
void set_work(void (*f)(void))
void set_work2(void* (*f)(void))

Устанавливает функцию f для запуска в отдельном потоке. Устанавливает функцию f для периодического запуска в главном потоке.

int run()

Запускается основной цикл с обработкой событий. Функция завершается когда закрывается последнее окно приложения.

void start()
Переводит приложение в режим "счет". Если был режим "редактирование", забирает значения из таблицы в соответствующие переменные. Запускает обновление таблицы по таймеру. Запускает расчетные функции. Если был режим "счет", предварительно вызывается stop.
void stop()
Переводит приложение в режим "редактирование". Если окно было в режиме "счет" больше не запускает расчетную функцию, если запущена функция в отдельном потоке выставляет флаг askedit=true и дожидается завершения потока. После этого обновляет таблицу и перестает обновлять ее по таймеру. Если окно в режиме "редактирование" — ничего не делает.
void ask_update()
Метит все виджеты как требующие обновления (например перерисовка для канваса).
void update()
Обновляет все виджеты требующие обновления. Обычно вызывается автоматически по таймеру.
void set_update_time(double time)
Устанавливает период проверки всех объектов на необходимость обновления в секундах. По умолчанию величина периода составляет 0.25 с.

3.2.4 Копирование экрана

unsigned char * grabbing(Fl_Window *, int x, int y, int dx, int dy)
unsigned char * grabbing(Fl_Window *);
void grabbing(Fl_Window *, int x, int y, int dx, int dy, const char * fname, int compr=COMPRESSION_DEFLATE)
inline void grabbing(Fl_Window *, const char * fname, int compr=COMPRESSION_NONE
inline void grabbing(Fl_Window &win, const char * fname, int compr=COMPRESSION_N
Копирует окно в буфер или файл. Для копирования в tiff файл поддерживается несколько компрессий COMPRESSION_NONE – без сжатия, COMPRESSION_DEFLATE – zip компрессия без потери качества (дает наилучший результат по сжатию и качеству изображения), COMPRESSION_JPEG – jpg компрессия с потерей качества (рекомендуется применять для сжатия плотно заполненных изображений).

3.2.5 Диалог выбора файла

char * file_dialog(const char * dialog_title=0, const char *pattern="All Fies (*.*)\0*.*\0", const char *fname="")
char * file_dialog_f(const char * message=0, const char *pattern=0, const char *fname=0)
Функции вызывают диалог выбора файла и возвращают строку с именем выбранного файла, если пользователь отказался от выбора файла – возвращается 0. file_dialog – найтивный win32 диалог, file_dialog_f – FLTK диалог.

Пожелания

Приглашаются волонтеры.

- Переписать отрисовку всех примитивов используя внешний растеризатор.
- Решить проблему синхронизации расчетного потока и интерфейса.

- Написать 3d canvas.
- Поддержка порта в msvc.
- Ваше предложение.

Благодарности людям участвовавшим в проекте. Голубенко Д.Ю. за ценные обсуждения, разработчикам библиотеки FLTK за удобный инструмент, Медведеву Д.А. за несколько "Shapes" и полезные обсуждения.

Интересно получить мнения об использовании mppg, пожелания о расширении возможностей, сообщения об ошибках (в коде и документации).

pru@hydro.nsc.ru

Прууэл Э. Р.