Каркасная библиотека Anti-GLUT
версия 0.0.05*

Справочное руководство

(C) Владимир Смирнов, 2006-2012
kkatarn@sleepgate.ru
smirnov@nocnt.ru

Автор не представляет никаких гарантий и не берет на себя никакой ответственности за корректность реализации каркасной библиотеки и за точность представленных ниже сведений.

Любая часть этого документа может быть воспроизведена на любом носителе без уведомления автора, при условии что будет дана ссылка на данный документ.

* Последнее обновление: 6 July 2012

1. Введение

Каркасная библиотека AntiGlut является интерфейсом прикладного уровня, позволяющим создавать OpenGL-приложения, переносимые между платформами Windows NT 4.0/2000/XP/Win7 и POSIX (FreeBSD/Linux/Solaris).

Библиотека сформирована посредством *обратного переноса с потерями* (*backport*) исходного кода иного программного проекта (*каркасной библиотеки* LibV), ее отладка протекает перманентно, по мере необходимости использования. Работоспособность проверена с несколькими примерами (три из которых требуют поддержки OpenGL версии 2.0).

1.1 Необходимость в каркасных библиотеках

Как известно[1], оригинальная спецификация графической системы OpenGL (далее – GL) определяет ее *интерфейс программирования графического оборудования ... предназначенный исключительно для взаимодействия с кадровым буфером.* GL не содержит никаких средств взаимодействия с периферийным оборудованием (в частности – с устройствами ввода). Более того, подготовка к GL к работе и завершение работы с GL требуют выполнения множества платформенно-зависимых действий. Привязка к платформе затрудняет переносимость; но даже в случае одной целевой платформы повторная реализация одного и того же кода (тривиально выполняемая по принципу *копировать-вставить*) нецелесообразна. Если же программист во что бы то ни стало желает реализовать все служебные функции самостоятельно (желание, весьма похвальное для студентов учебных заведений), то и здесь каркасная библиотека может быть полезной – по крайней мере, в качестве источника идей реализации.

На данный момент существует большое количество каркасных библиотек, изолирующих платформенно-зависимый код. Стандартом де-факто стала OpenGL utility toolkit (GLUT). Она используется, в частности, при подготовке справочных руководств по программированию и многочисленных учебных изданий, посвященных GL. Однако (несмотря на справедливое в свое время замечание Марка Киллгарда о том, что служебный код инициализации Direct3D имеет на порядок больший размер, нежели код инициализации GL) к настоящему времени (GLUT 3.7) дерево исходного кода GLUT и сопутствующих ресурсов занимает почти 10 Мб (а bzip2-архив – около 3 Мб). Это по меньшей мере затрудняет использования исходного кода в процессе анализа реализаций платформенно-зависимого кода.

В действительности на практике все функции GLUT используются редко; например, для «русскоязычного» приложения нет никакой пользы от некириллической шрифтовой поддержки GLUT (вообще, вывод текста в кадровый буфер – отдельная больная тема, универсального и «хорошего» способа решения которой попросту нет).

1.2 Реализация AntiGlut

По сравнению с другими каркасными библиотеками аналогичного назначения AntiGlut имеет существенно меньший размер. Несмотря на то, что исходный код получен обратным переносом, он достаточно структурирован (по крайней мере, автору так кажется). Вся библиотека сосредоточена внутри одного файла `antiglut.c`. Поэтому если некоторый файл `foo.c` полностью содержит код программы, использующей AntiGlut, то для сборки достаточно сказать (в предположении о «нормальности» разработчика – платформа WinAPI):

```
cl foo.c antiglut.c -D__WIN32__ opengl32.lib <...>
```

либо для POSIX-платформы:

```
gcc foo.c antiglut.c -D__UNIX__ -lGL <...>
```

В зависимости от кода может потребоваться компоновка с дополнительными библиотеками (см. Makefile-ы примеров; на POSIX-платформах нужен GNU make). На WinAPI-платформах требуется, по меньшей мере, подключение библиотек `kernel32.lib`, `user32.lib` и `gdi32.lib`.

Основным отличием AntiGlut от стандарта де-факто – GLUT – является парадигма обработки сообщений. Если GLUT использует механизм обратного вызова, то AntiGlut реализует X11-подобную схему опроса очереди сообщений. Поэтому цикл обработки должен быть

[1] www.oss.sgi.com (оказаться полезным может также официальный портал GL www.opengl.org)

реализован в клиентской части кода (см. примеры; UPD: в libv-core/source/agmainloop.c есть agMainLoop()).

Загрузка расширений OpenGL не поддерживается. Однако если драйвер GPU поддерживает функциональность, определенную стандартами выше GL 1.1, то при инициализации эту функциональность (до версии GL 2.0 – работа с вершинными и пиксельными шейдерами) можно затребовать. Как исключение – поддерживается функциональность геометрических шейдеров (расширение GL_EXT_geometry_shader4).

Прототипы всех функций OpenGL уже содержаться в файле `antiglut.h`, поэтому включать системный файл `GL/gl.h` не нужно (и нельзя – это приведет к ошибке на этапе компиляции).

AntiGlut использует глобальные переменные, поэтому создавать можно только одно окно на процесс (если семантика вызовов устраивает, но нужно одно окно на поток – обращайтесь к библиотеке LibV, но учтите – LibV не просто кривая, а _очень кривая_ :*)).

2. Структуры данных

Функции, непосредственно связанные с управлением подсистемой GL, немногочисленны и принимают только скалярные параметры. Определение структур данных требуется только для работы с оконной подсистемой.

Основной структурой данных, с которой работают функции опроса очереди сообщений, является структура `ui_event_t`, определенная в файле `antiglut.h` следующим образом:

```
typedef struct ui_event_s
{
      int         type; /* тип события */
      GLfloat     time; /* время получения в секундах, от первого полученного */
      int         client_x, client_y; /* клиентские координаты курсора мыши */
      int         screen_x, screen_y; /* экранные координаты курсора мыши */
      int   client_dx, client_dy; /* смещение курсора от последнего события мыши */
      unsigned button; /* битовая маска нажатых клавиш мыши */
      unsigned state; /* битовая маска клавиатурных модификаторов */
      ui_key_event_t key; /* данные для события нажатия/отпускания клавиши */
      ui_configure_event_t configure; /* данные для события - состояния окна */
}     ui_event_t, *ui_event_p;
```

Поле `type` имеет одно из значений, приведенных в табл. 1.

Таблица 1. Возможные типы событий

Имя	Значение
`ui_key_event`	Событие от клавиатуры (нажатие или отпускание клавиши) либо событие от кнопки или колеса мыши; дополнительные данные содержатся в поле `key`.
`ui_pointer_motion_event`	Перемещение курсора мыши.
`ui_window_focus_in_event`	Окно принимает фокус ввода; дополнительная информация содержится в поле `configure`.
`ui_window_focus_out_event`	Окно теряет фокус ввода.
`ui_window_expose_event`	Клиентская область требует перерисовки.
`ui_window_map_event`	Окно стало видимым.
`ui_window_unmap_event`	Окно стало невидимым.
`ui_window_move_event`	Окно перемещено.
`ui_window_size_event`	Окно изменило размер.
`ui_wm_event`	Сообщение от подсистемы управления окнами (в Иксовой терминологии – от менеджера окон).

Помимо приведенных выше допустимых значений в отладочных целях определены `ui_no_event` и `ui_max_event` (целочисленный код последнего заведомо больше кода любого из определенных выше).

Поле `time` содержит относительное время получения в сообщения секундах. После (и *только* после) инициализации гарантируется близость начала отсчета для этого поля с началом отсчета времени функцией **agTickCount**() (следует помнить, что последняя возвращает время в *миллисекундах*).

Два следующих поля – `client_x` и `client_y` – содержат координаты курсора мыши относительно клиентской области окна. Поля – `screen_x` и `screen_y` – содержат координаты курсора мыши относительно верхнего левого угла экрана (в реальном приложении они практически бесполезны и использовать их *не* рекомендуется; более того, они могут быть удалены в последующих версиях AntiGlut). Поля `client_dx` и `client_dy` содержат приращения координат от предпоследнего к последнему перемещению мыши. Поле `button` содержит битовую маску нажатых клавиш мыши. Битовые маски отдельных клавиш приведены в табл. 2.

Таблица 2. Определенные битовые маски клавиш мыши

Имя	Соответствующая клавиша
`UI_LBUTTON_MASK`	Левая.
`UI_MBUTTON_MASK`	Средняя (или нажатие на колесо прокрутки).
`UI_RBUTTON_MASK`	Правая.

Поле state содержит битовую маску клавиш-модификаторов (табл. 3).

Таблица 3. Определенные битовые маски клавиатурных модификаторов

Имя	Соответствующая клавиша
`UI_NUM_LOCK_MASK`	Num Lock
`UI_CAPS_LOCK_MASK`	Caps Lock
`UI_SCROLL_LOCK_MASK`	Scroll Lock
`UI_SHIFT_MASK`	Shift (хотя бы один)
`UI_CONTROL_MASK`	Control (хотя бы один)
`UI_ALT_MASK`	Alt (хотя бы один)
`UI_SUPER_MASK`	Win (хотя бы один) (прим.: польза – только на POSIX-платформах; под WinAPI эта клавиша *не* блокируется)

Для всех событий, кроме `ui_pointer_motion_event`, доступна дополнительная информация. В случае события `ui_key_event` устанавливается поле `key`, определенное в `antiglut.h`:

```
typedef struct ui_key_event_s
{
      GLboolean press;  /* равно true если клавиша нажата */
      unsigned scan;    /* специфичный для AntiGlut скан-код; см. antiglut.h */
      unsigned acode;   /* ASCII код клавиши; верх таблицы зависит от локали */
      unsigned ucode;   /* Юникод клавиши (хост-порядок байт) */
      /* Код UTF-8 */
      unsigned utf8_len;
      unsigned char utf8[AG_UTF8_MAXLEN];
}     ui_key_event_t, *ui_key_event_p;
```

Для остальных событий устанавливается поле `configure`:

```
typedef struct ui_configure_event_s
{
      GLboolean focused;      /* окно имеет фокус ввода */
      GLboolean visible;      /* окно видимо */
      /* прямоугольник перерисовки */
      unsigned exposed_left, exposed_top, exposed_right, exposed_bottom;
      int x, y;               /* левый верхний угол окна  */
      unsigned width, height; /* размер клиентской части окна */
      /* window manager request */
```

```
    unsigned wm_event;      /* сообщение от менеджера окон */
}   ui_configure_event_t, *ui_configure_event_p;
```

Для поля `configure.wm_event` определено единственное значение `ui_wm_close`; в это значение поле устанавливается в ответ на интерактивное удаление окна (нажатие кнопки закрытия или комбинации Alt+F4 на WinAPI-платформе).

3. Функции

3.1 Инициализация и завершение работы

Перед первым вызовом любой команды GL библиотека AntiGlut должна быть сконфигурирована. Инициализация выполняется последовательным вызовом функций **agWindowConfig()** (или **agWindowedConfig()**) и **agInit()**. Первые две

```
GLvoid
agWindowConfig(char *title,     /* заголовок окна */
      GLuint width,             /* ширина клиентской части окна */
      GLuint height,            /* высота клиентской части окна */
      GLboolean fullscreen);    /* требуется полноэкранный режим */

GLvoid
agWindowedConfig(char *title, /* заголовок окна */
      GLuint width,             /* ширина клиентской части окна */
      GLuint height,            /* высота клиентской части окна */
      GLuint left,              /* положение левого ... */
      GLuint top);              /* ... верхнего угла окна */
```

служат для указания геометрии окна вывода, а вторая

```
GLboolean
agInit(GLuint required_gl_version,    /* требуемая версия OpenGL */
      GLboolean depth,          /* необходим буфер глубины */
      GLboolean stencil,        /* необходим буфер трафарета */
      GLboolean dst_alpha);     /* необходим альфа-канал в буфере кадра */
```

служит для инициализации подсистемы GL.

Вызов одной из первых двух функций не является обязательным. Если он опущен, то создается окно на весь экран (заголовок отсутствует), положение окна изменять нельзя. Вызов функции **agWindowedConfig()** указывает на необходимость создания окна, размеры которого можно изменять (создание полноэкранного окна и изменение разрешения экрана не допускается).

Функция **agInit()**, помимо инициализации GL, предназначена для установки формата буфера кадра. Если приложению необходим z-буфер, то в качестве второго параметра следует указать GL_TRUE; аналогично следует поступить, если требуются буферы трафарета и альфа-канала. Если реализация не в состоянии обеспечить требуемый уровень (версию) GL или не поддерживает требуемые буферы, то вызов завершится с ошибкой (GL_FALSE); в случае успешного завершения возвращается GL_TRUE.

Параметр required_gl_version может принимать одно из следующих значений (указанные ниже макроподстановки, как и большинство других, определены в `antiglut.h`):

`AG_REQUIRED_1_1` – требуется только базовая функциональность (GL версии 1.1);

`AG_REQUIRED_1_2` – требуется функциональность GL версии 1.2 (отсутствие поддержки функциональности, соответствующей расширению GL_ARB_imaging, не считается ошибкой инициализации);

`AG_REQUIRED_1_3` – требуется функциональность GL версии 1.3;

`AG_REQUIRED_1_4` – требуется функциональность GL версии 1.4;

`AG_REQUIRED_1_5` – требуется функциональность GL версии 1.5 (отсутствие поддержки функциональности, соответствующей расширению ARB_occlusion_query, не считается ошибкой инициализации);

`AG_REQUIRED_2_0` – требуется функциональность GL версии 2.0 (на этом уровне предпринимается попытка загрузки функциональности расширения GL_EXT_geometry_shader4).

После успешного завершения вызова **agInit()** клиентскому коду доступны все функции, регламентированные соответствующим стандартом OpenGL (если GPU не поддерживает соответствующую функциональность, то вызов завершится ошибкой).

Функция **agInit()** всегда создает окно с двойной буферизацией.

`GLvoid` **agShutdown**();

Функция должна вызоваться клиентской частью по завершению работы с GL.

3.2 Работа с очередью сообщений

`GLboolean` **uiWait(**`void`**);**

Ожидает сообщений (блокирующий вызов). Возвращает GL_TRUE при отсутствии ошибки.

`GLboolean` **uiPending(**`void`**);**

Возвращает GL_TRUE, если очередь сообщений непуста; если сообщений нет, возвращает GL_FALSE. Неблокирующий вызов.

`GLboolean` **uiNextEvent(**`ui_event_p ev_return`**);**

Извлекает из очереди и помещает по адресу `ev_return` очередное сообщение. Возвращает GL_TRUE при отсутствии ошибки. Блокирующий вызов (если сообщения нет, ожидает его).

`ui_event_p` **uiPeekEvent(**`unsigned offset_from_first_push`**);**
`ui_event_p` **uiPeekLastEvent(**`unsigned offset_from_last_push`**);**

Неблокирующие вызовы; не модифицируют очередь.

`GLboolean` **uiRemoveEvent(**`unsigned offset_from_first_push, ui_event_p ev_return`**);**
`GLboolean` **uiRemoveLastEvent(**`unsigned offset_from_last_push, ui_event_p ev_return`**);**

Неблокирующие вызовы; модифицируют очередь. Эквивалент пары **uiPending()**+**uiNextEvent()**.

3.3 Вспомогательные процедуры

`void` **uiAutoRepeat(**`GLboolean enable`**);**

Управление режимом автоповтора нажатий клавиш (по умолчанию автоповтор включен). Вызов **uiAutoRepeat(**`GL_FALSE`**)** блокирует автоповтор.

`void` **uiWarpPointer(**`unsigned client_x,unsigned client_y`**);**

Принудительное изменение положения курсора мыши.

`void` **uiAutoWarp(**`GLboolean enable`**);**

Управление автотрекингом курсора мыши. Вызов **uiAutoWarp(**`GL_TRUE`**)** приводит к сохранению текущих координат курсора, скрытию указателя и началу автотрекинга. В процессе автотрекинга клиентские координаты курсора могут принимать любые значения между `-MAXLONG` и `MAXLONG`; экранные координаты использовать нельзя. Предпочтительно не использовать и клиентские координаты, пользуясь лишь инкрементами (поля `client_dx` и `client_dy` структуры `ui_event_t`) от последнего события мыши.

`void` **agWindowTitle(**`char *title`**);**

Смена текста в заголовке окна.

`GLuint` **agTickCount(**`void`**);**

Возвращает значение счетчика миллисекунд. Начало отсчета произвольно, но близко к началу отсчета поля `time` события (поле `time` содержит время в секундах).

`GLvoid` **agSleep(**`GLuint milliseconds`**);**

Отдает ЦП другому потоку как минимум на `milliseconds` миллисекунд (задержка выполнения).

4. Сборка демонстрационных программ

Makefile-ы в процессе. Но, в целом, под тремя основными юниксами собрать можно без проблем (нужен GNU make, в планах – нативные мэйкфайлы для FreeBSD). И забыта макось. Совсем.

5. Баги и прочая

Нужна внутренняя чистка, после которой надо сделать обратное портирование антиглута в дерево кода LibV. Но сначала нужно переписать ядро LibV. Это до пенсии)))

Под солярисом нету расширения xf86VidMode, разрешение экрана не сменить; но под солярисом вообще много чего нету, так что это не столь важно. А вообще, давно следует прикрутить расширение RandR. И загружать libGL.so и все расширения иксов динамически. И довести до OpenGL четвертой версии в обеих профилях. А можно заменить вызовы Xlib на XCB. Но проще умереть.