

OpenGL и DirectX: программирование графики. Для профессионалов (+CD)

А. Евченко

Задачи, связанные с различными аспектами компьютерной графики, возникают в самых различных сферах применения информационных технологий. Иногда они являются основой системы, как в играх и некоторых тренажерах, чаще играют важную вспомогательную роль, как в системах обучения, моделирования, управления технологическими процессами. Невозможно представить себе разработку компьютерной графики без использования технологий OpenGL и DirectX. В этой книге максимально подробно описываются обе эти технологии.

Помимо рассмотрения плоской и трехмерной графики, отдельная глава посвящена проблемам создания стереоизображений. Эта перспективная модель графики до сих пор не рассматривалась так подробно, и данная книга ликвидирует этот пробел.

На прилагаемом компакт-диске находится код всех приложений, которые рассматривались в качестве примеров.

Содержание

Введение

От издательства

Часть 1. Программирование для Windows

Глава 1. Создание простейшего приложения

Параметры функции WinMain

Функция MessageBox()

Назначение и структура сообщений

Полноценное оконное приложение

Некоторые полезные функции и структуры данных

Типы данных Windows

Глава 2. Обмен сообщениями

Глава 3. Элементы компьютерной графики

Вывод простейших графических примитивов

Вывод динамических изображений

Формирование статических изображений

Глава 4. Измерение временных интервалов

Задания для самостоятельного выполнения

Часть 2. Средства взаимодействия с пользователем в среде Windows

Глава 5. Создание и использование меню

Включение меню в окно приложения

Порядок обмена сообщениями при работе с меню

Управление состоянием пунктов меню

Средства модификации и построения меню

Глава 6. Дочерние окна

Ввод строки пароля в дочернем окне

Графические элементы управления

Создание подклассов

Элемент управления Edit

Глава 7. Диалоговые окна

Типы диалогов

Простейший модальный диалог

Немодальный диалог

Глава 8. Элементы управления в диалоговых окнах

Окна редактирования и статический текст

Применение переключателей

Глава 9. Ввод иероглифов

Глава 10. Скелетный проект

Задания для самостоятельного выполнения

Часть 3. Средства GDI-графики

Глава 11. Контекст устройства

Назначение и состав контекста

Создание элементов контекста

Глава 12. Изменение элементов контекста

Последовательность действий при изменении контекста

Изменение контекста и скорость рисования

Глава 13. Вывод графических примитивов

Прямоугольник со скругленными углами

Многоугольные фигуры

Эллипс и его фрагменты

Кривые Безье

Точки

Глава 14. Режимы вывода

Цвет текста

Фоновый цвет примитивов

Глава 15. Запрос параметров контекста

Глава 16. Контекст памяти и двойная буферизация

Копирование участков экрана

Рисование в системной памяти средствами GDI

Двойная буферизация

Глава 17. Работа с растровыми изображениями

Хранение графических данных в формате BMP

Загрузка данных из BMP-файла

Вывод растровых изображений на экран

Непосредственный доступ к графическим данным

Запись изображений в файл

Глава 18. Текстовые сообщения и шрифты

Глава 19. Атрибуты контекста устройства и текста

Глава 20. Создание и применение регионов

Типы регионов

Работа с окнами произвольной формы

Задания для самостоятельного выполнения

Часть 4. Трехмерная графика

Глава 21. Предварительные сведения

Основные термины и определения

Графический конвейер

Структура видеобuffers

Переключение страниц

Стабилизация периода переключения страниц

Структура модели виртуального мира

Свойства граней

Описание вершин грани

Структуризация объектов

Типы объектов модельного мира

Системы координат графических библиотек

Этапы работы графического конвейера

Глава 22. Инициализация библиотек и вывод примитивов

Простейшее приложение OpenGL

Инициализация библиотеки OpenGL

Вывод треугольников

Кадрирование изображения в OpenGL

Использование расширений

Измерение временных характеристик

Доступ к расширениям

Рисование в битовую карту DIB-формата

Настройка конвейера

Удаление тыльных граней

Удаление конкурирующих точек

Имитация полупрозрачности

Типы примитивов

Материал и освещение

Наложение текстур в OpenGL

Глава 23. Применение библиотеки DirectX

Характеристика COM-объектов

Инициализация DirectX

Поля структуры PIXELFORMATDESCRIPTOR

Инициализация библиотеки DirectX

Вывод треугольников

Кадрирование изображения библиотекой DirectX

Настройка конвейера

Удаление тыльных граней

Удаление конкурирующих точек

Отсечение по видимому объему

Настройка кадрирования

Задание освещенности

Глава 24. Геометрические преобразования

Преобразования на плоскости

Преобразования в однородных координатах

Преобразования в пространстве

Связь однородных и декартовых координат

Перенос и масштабирование модели объекта

Вращение объектов вокруг координатных осей

Проецирование модели на плоскость

Приведение пирамиды видимости к каноническому объему

Отображение трехмерной сцены

Отображение средствами DirectX

Отображение средствами OpenGL

Глава 25. Формирование стереоизображений

Общие сведения

Обзор демонстрационных примеров

Задания для самостоятельного выполнения

Список литературы

Алфавитный указатель

Введение

Задачи, связанные с применением компьютерной графики, возникают в самых различных сферах информационных технологий. Иногда они являются основой системы, как в играх и некоторых тренажерах, но чаще играют важную вспомогательную роль в системах обучения, моделирования и управления технологическими процессами. Интерес к этим задачам традиционно высок. Почти каждый студент-первокурсник, выбравший компьютерную специальность, в глубине души уверен, что его профессия позволит всю оставшуюся жизнь если не играть за компьютером, то, по крайней мере, проектировать компьютерные игры.

Лучше всего построить книгу таким образом, чтобы изучение графики не требовало ни предварительных знаний, выходящих за пределы школьного курса информатики, ни привлечения дополнительной литературы. Книга и компьютер под рукой — этого достаточно, чтобы освоить разработку систем, активно использующих методы компьютерной графики. Но при этом пришлось бы начинать очень уж издалека, рискуя не добраться до графики. Компромиссное решение заключается в том, что у читателя не предполагается никаких начальных навыков программирования для Windows, необходимый минимум дан в первых главах. Но нужно иметь опыт программирования и отладки программ на языке C++ хотя бы в среде MS DOS.

Книга состоит из нескольких частей. В первой части изложен материал, позволяющий начинающим освоить создание полноценных оконных приложений, осмыслить принятую в операционной системе организацию вычислительного процесса, познакомиться с элементами API (Application Program Interface), которые обеспечивают взаимодействие ОС Windows с прикладной программой. Вторая и третья части посвящены решению задач плоской (2D) графики средствами стандартного графического интерфейса GDI (Graphic Device Interface).

Далее при изучении трехмерной графики приводятся реализация и сравнительный анализ решения типовых задач формирования трехмерных (3D) и стереоизображений средствами библиотек DirectX и OpenGL.

В данной книге нет подробного рассмотрения фундаментальных алгоритмов компьютерной графики и их программной реализации. Эти алгоритмы уже реализованы библиотеками и аппаратурой графических ускорителей, поэтому дается общее представление об их работе, необходимое для взаимодействия с графическими библиотеками. Было время, когда редкая книга по вычислительной технике обходилась без описания работы сумматора и алгоритмов ускоренного умножения. Но поскольку машинные команды даны нам как нечто неизменное и атомарное, стало понятно, что детали построения арифметико-логических устройств нужны весьма узкому кругу программистов. Аналогичный процесс происходит сейчас и в области компьютерной графики.

В коротких примерах, демонстрирующих возможности API, GDI, DirectX и других программных интерфейсов, автор сознательно избегал объектно-ориентированного подхода. Разработка собственной системы классов, инкапсулирующих вызовы изучаемых интерфейсов, упрощает работу с ними. Но при этом читатель отклоняется от деталей реализации самой графики. Целью же книги является максимально подробное исследование возможностей создания и обработки графических изображений. Объектно-ориентированное программирование использовалось только в нескольких более объемных проектах.

Автор надеется, что книга будет полезна как профессиональным программистам, осваивающим новую для себя область, так и начинающим пользователям в учебном процессе. Преподаватель найдет здесь обширный материал для формирования индивидуальных заданий по курсовому проектированию и лабораторным работам, а студент — хорошее пособие для выполнения этих заданий.

Глава 7. Диалоговые окна

Типы диалогов

Рассмотренные выше графические управляющие элементы чаще всего размещают в диалоговом окне. Вызовы `CreateWindow()` для управляющих элементов в этом случае выполняются функцией вывода диалогового окна автоматически. Пример диалогового окна с двумя кнопками, создаваемого в проекте `DlgMin`, показан на рис. 7.1.

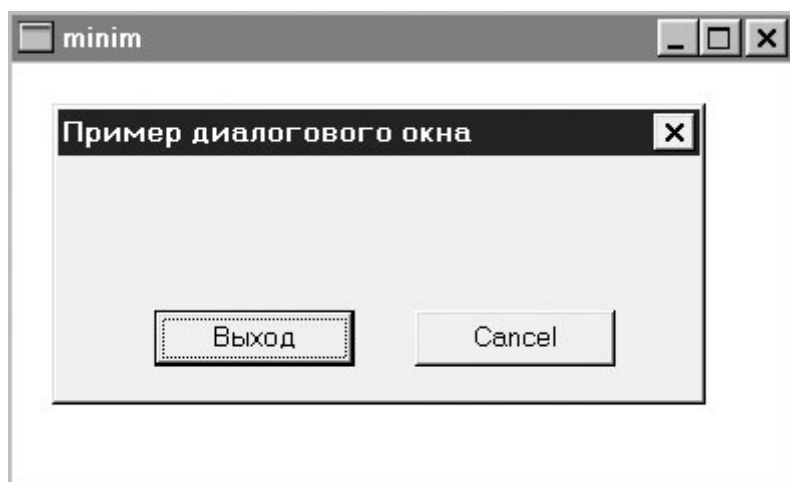


Рис. 7.1. Диалоговое окно

Шаблон диалога задается, как и меню, в файле ресурсов. Диалоговое окно выводится на экран функциями `DialogBoxParam()` и `CreateDialogParam()`. Первая функция создает модальный диалог, который переключает обработку сообщений от внешних устройств на собственный цикл опроса очереди и до закрытия диалога не позволяет переместить щелчком мыши фокус ввода на основное окно. Вторая функция создает немодальный диалог, который получает сообщения от мыши из основного цикла опроса очереди. Щелчком мыши пользователь может переключать фокус ввода между основным окном и немодальными диалоговыми окнами.

Шаблон показанного на рисунке диалогового окна задается в файле ресурсов, как показано в листинге 7.1.

Листинг 7.1

```

DIALOGMIN DIALOG 10, 10, 160, 60
STYLE DS_MODALFRAME| WS_POPUP| WS_VISIBLE| WS_CAPTION| WS_SYSMENU
CAPTION "Пример диалогового окна"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON    "OK", IDOK, 24, 38, 50, 14
    PUSHBUTTON       "Cancel", IDCANCEL, 89, 38, 50, 14
END

```

В первой строке указываются присвоенный программистом идентификатор ресурса (DIALOGMIN), тип ресурса (DIALOG) и положение окна в рабочей области. После этого указываются уже известные нам флаги стиля окна, текст области заголовка, размер и тип шрифта. Строками, заключенными между BEGIN и END, перечисляются дочерние окна, которые будут размещены в окне диалога. В данном случае это два окна класса button с названиями OK, Cancel и идентификаторами IDOK, ID2. Константа ID2 будет помещена редактором ресурсов в файл resource.h, а IDOK — нет, так как она уже определена в файле winuser.h.

Указанный текст нет смысла вставлять в файл ресурсов при помощи текстового редактора, так как редактор ресурсов предоставляет визуальную технологию для описания диалогового окна и размещения на нем графических управляющих элементов. Редактор ресурсов позволяет сформировать внешний вид диалога, размещая в диалоговом окне изображения управляющих элементов. Для каждого элемента можно вызвать окно свойств, определяющих его поведение. Например, обычная кнопка имеет в шаблоне тип PUSHBUTTON, но при задании свойства Default Button кнопка с идентификатором IDOK задается следующей строкой:

```
DEFPUSHBUTTON    "OK", IDOK, 24, 38, 50, 14
```

Это означает, что по умолчанию в окне этой кнопки находится фокус ввода. Пользователь может переключать фокус ввода щелчком мыши или нажатием клавиши Tab. Циклическая передача фокуса по нажатии этой клавиши происходит только между элементами, у которых в редакторе ресурсов свойство Tabstop установлено в состояние true. В шаблоне диалога реакция на табуляцию задана по умолчанию. При ее отмене в описании элемента появляется дополнительный флаг, как это показано в следующем определении:

```
DEFPUSHBUTTON    "OK", IDOK, 24, 38, 50, 14, NOT WS_TABSTOP
```

Оконный класс дочернего окна в описаниях кнопок явно не указан. В строке, описывающей элемент управления для ввода текста, указываются тип EDITTEXT, идентификатор IDC_EDIT1 и координаты элемента в окне. Название оконного класса edit определяется типом элемента:

```
EDITTEXT        IDC_EDIT1, 53, 14, 21, 9
```

Для новых элементов управления, описание которых начинается со слова CONTROL, принят формат описания с явным указанием оконного класса:

```
CONTROL         "<Название>", IDC_TREE1, "SysTreeView32", 51, 7, 24, 14
```

Такой формат позволяет прикладному программисту создавать собственные управляющие элементы, оформляя их в виде COM-объектов. В данном случае вместо имени оконного класса указывается программное имя или глобально-уникальный идентификатор (GUID) объекта:

```
CONTROL         "<Название>", IDC_MSFLXGRID,
                "{6262D3A0-531B-11CF-91F6-C2863C385E30}",
                WS_TABSTOP, 7, 44, 180, 69
```

Модальный диалог создается по показанному выше шаблону DIALOGMIN при помощи функции int DialogBoxParam(GetModuleHandle(NULL), MAKEINTRESOURCE(DIALOGMIN), hWnd, (DLGPROC) DialogMin, 17). Ее можно вызвать из основного окна. Функция не возвращает управления, пока выведенное на экран диалоговое окно не будет уничтожено вызовом функции EndDialog(). При завершении функция возвращает код выхода.

Параметры функции DialogBoxParam() задают дескриптор приложения, идентификатор ресурса, дескриптор основного окна и имя оконной функции. Последний параметр (в данном примере его значение равно 17) передается оконной процедуре диалога в поле lParam сообщения WM_INITDIALOG. Он используется редко, поэтому вместо функции DialogBoxParam() можно применять макрос DialogBox (GetModuleHandle(NULL), MAKEINTRESOURCE(DIALOGMIN), hWnd, (DLGPROC) DialogMin).

Для создания немодального диалога тоже есть макрос и функция, которые имеют те же параметры, но возвращают дескрипторDlgWin диалогового окна:

```
HWND DlgWin = CreateDialog(GetModuleHandle(NULL),  
MAKEINTRESOURCE(DIALOGMIN), hWnd, (DLGPROC) DialogMin);
```

При вызове этой функции на экране появится немодальное диалоговое окно. При завершении немодального диалога окно следует уничтожить вызовом функции DestroyWindow().

Простейший модальный диалог

Из оконной процедуры приложения модальное диалоговое окно можно вызвать в ответ на щелчок левой кнопкой мыши, как это показано в листинге 7.2.

Листинг 7.2

```
case WM_LBUTTONDOWN:  
    DialogBox (GetModuleHandle(NULL),  
              MAKEINTRESOURCE(DIALOGMIN), hWnd, (DLGPROC) DialogMin);  
return 0;
```

Оконная функция диалога, которая в дальнейшем изложении будет называться диалоговой процедурой, похожа на оконную функцию основного окна. Но есть и несколько отличий. Перед отображением диалога в диалоговую процедуру поступит сообщение WM_INITDIALOG. Диалоговая процедура не выполняет явного вызова обработки сообщений по умолчанию DefWindowProc(), но неявно такая обработка выполняется и использует значение, возвращаемое диалоговой процедурой. Если диалоговая процедура возвращает значение false, то дополнительная обработка производится. Некоторые источники выделяют особое положение сообщения WM_INITDIALOG, указывая, что для него дополнительная обработка включается возвратом значения true.

Следует помнить, что размещенные в диалоге графические элементы управления являются дочерними окнами, а потому диалоговая процедура извещается о событиях в этих окнах сообщениями WM_COMMAND.

Для открытия окна в системе уже должен быть зарегистрирован соответствующий оконный класс. Оконные классы органов управления, унаследованных от Windows 3.1, система регистрирует автоматически. Для новых элементов следует включить в функцию InitApp() вызов InitCommonControls(), который произведет регистрацию классов, а в список компонуемых библиотек добавить comctl32.lib.

Текст оконной функции диалога, изображенного на рис. 7.1 находится в файле DialogMin.cpp проекта DlgMin и приведен в листинге 7.3.

Листинг 7.3

```

#include <windows.h>
#include "resource.h"
HWND DlgWin = 0; //Дескриптор диалогового окна в немодальном режиме
BOOL DialogMin(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
switch(message)
{
case WM_INITDIALOG: //По этому сообщению изменяется заголовок диалога
SetWindowText(hDlg, "Простейший диалог");
//При появлении окна выводится текст в поле заголовка
SetWindowText(GetDlgItem(hDlg, IDOK), "Выход");
//Изменить надпись на кнопке IDOK
//SetFocus(GetDlgItem(hDlg, IDCANCEL));
//Передача фокуса кнопке IDCANCEL
return TRUE;
case WM_COMMAND: //Обработка сообщений от управляющих элементов.
//В lParam в этот момент находится дескриптор окна элемента,
//в LOWORD(wParam) - идентификатор этого элемента,
//а в HIWORD(wParam) - код извещения.
//Чтобы закрыть диалог, надо вызвать функцию EndDialog().
if (LOWORD(wParam) == IDOK)
{EndDialog(hDlg, 0); return TRUE;}
//По второй кнопке не выполняем никаких действий:
if (LOWORD(wParam) == ID2) return TRUE;
break;
case WM_CLOSE:
{EndDialog(hDlg, 1);
return TRUE;
}
}
return FALSE;
}

```

Если в последней строке использовать оператор `return TRUE`, то обработка сообщений по умолчанию будет выполняться, но программист сообщает обрабатывающей функции, что он уже сделал все необходимое самостоятельно. В результате возврата `TRUE` не обработается по умолчанию сообщение `WM_ERASEBKGD`. В результате не будет отрисован фон окна диалога, и пользователь увидит только графические элементы управления.

Теперь нужно добавить в основную функцию приложения обработку сообщения `WM_ENTERIDLE`. Диалоговое окно не формирует его явно, и сообщение посылается при обработке по умолчанию. Легко убедиться в том, что сообщение приходит от диалогового окна всегда, то есть диалоговая процедура возвращает управление не в операционную систему, а в обработчик по умолчанию.

Следует удалить комментарии в следующей строке при обработке сообщения `WM_INITDIALOG`:

```
//SetFocus(GetDlgItem(hDlg, IDCANCEL));
```

Чтобы при открытии диалога фокус ввода был действительно передан кнопке `IDCANCEL`, необходимо завершить обработку оператором `return FALSE`, то есть и в ветви `case WM_INITDIALOG` объем дополнительной обработки увеличивается при возврате `FALSE`.

Диалоговое окно не является дочерним по отношению к окну, из которого оно создано. Но окно-создатель является собственником диалога. По известному дескриптору диалога `hDlg` можно получить дескриптор владельца функцией `GetParent(hDlg)`. Напомним, что эта же функция используется для получения родительского окна по дескриптору дочернего.

При первом знакомстве с дочерними окнами у многих возникает вопрос, нужен ли окну числовой идентификатор, если оно однозначно определяется дескриптором `HWND`? Я думаю, основная причина заключается в том, что значение идентификатора задает программист, а значение дескриптора окна дает операционная система и оно может изменяться в зависимости от ситуации в системе. На примере диалоговых окон можно увидеть, как идентификатор позволяет полностью изолировать приложение от дочернего окна. Для размещенных в диалоге графических элементов управления программист не пишет явно операторов открытия окна и все взаимодействие основано на обмене сообщениями.

Немодальный диалог

При использовании одного шаблона модальный и немодальный диалоги можно реализовать общей оконной функцией, но такая функция должна учитывать различия в организации

завершения модального и немодального диалогов. В проекте Два_Диалога по щелчку левой кнопкой мыши оконная процедура запускается как модальный диалог с дескриптором DlgMod:

```
DialogBoxParam(GetModuleHandle(NULL),  
MAKEINTRESOURCE(DIALOGMIN), hWnd, (DLGPROC) DialogMin, 17);
```

По щелчку правой кнопкой вызывается немодальный диалог с дескриптором DlgWin:

```
DlgWin = CreateDialog(GetModuleHandle(NULL),  
MAKEINTRESOURCE(DIALOGMIN), hWnd, (DLGPROC) DialogMin);
```

Диалоговая процедура Dialog реализована в файле Dialog.cpp проекта, как показано в листинге 7.4.

Листинг 7.4

```
#include <windows.h>  
#include "resource.h"  
HWND DlgWin; //Дескриптор немодального диалогового окна.  
HWND DlgMod; //Дескриптор модального диалогового окна.  
HWND ButtonWnd; //Дескриптор кнопки в окне диалога.  
BOOL Dialog (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)  
{  
    switch(message)  
    {  
        case WM_INITDIALOG:  
            //По этому сообщению изменяется заголовок диалогового окна. Типы диалога  
            //различаются по значению 17 параметра lParam  
            if(lParam==17) //При появлении окна вывести текст в поле заголовка.  
            {  
                SetWindowText(hDlg, "Модальный диалог");  
                DlgMod = hDlg;  
            }  
            else SetWindowText(hDlg, "Немодальный диалог");  
            ButtonWnd = GetDlgItem(hDlg, IDOK);  
            return TRUE;  
            //При щелчке мышью в рабочей области диалогового окна выполняется обработка  
            //сообщения WM_LBUTTONDOWN:  
        case WM_LBUTTONDOWN:  
            if(hDlg==DlgWin)  
                MessageBox(hDlg, "Немодальный", "Диалог", NULL);  
            else  
                MessageBox(hDlg, "Модальный", "Диалог", NULL);  
            return TRUE;  
        case WM_CLOSE:{  
            EndDialog(hDlg, TRUE);  
            //if(hDlg==DlgWin)DlgWin=0;  
            return TRUE;  
        }  
        case WM_COMMAND: //Обработка сообщений от управляющих элементов.  
            if (LOWORD(wParam)==IDOK )  
                {EndDialog(hDlg, TRUE);  
                //if(hDlg==DlgWin)DlgWin=0;  
                return TRUE;  
            }  
            if (LOWORD(wParam)==ID2 ) {return TRUE; break;}  
            return FALSE;  
    }  
}
```

Следует еще раз отметить, что отдельные очереди создаются не к разным окнам, а к разным потокам. Используя функцию GetWindowThreadProcessId() легко убедиться, что окна подменю, диалоговое и основное окно связаны с одним потоком. То есть модальный диалог не переключает сообщения на свою очередь, а подавляет поступление щелчков мыши в основное окно.

Также следует обратить внимание на закомментированный оператор if(hDlg == DlgWin) DlgWin = NULL. Увеличение размера системных программ привело к резкому обострению ситуации с надежностью программного обеспечения. В настоящее время значительная часть усилий, направленных на повышение надежности, перенесена с тестирования и выявления ошибок на написание более надежного программного кода. В кодах коммерческих продуктов можно увидеть тексты, в которых объявляется нулевая локальная переменная, а строкой ниже она же проверяется условным оператором на равенство нулю. А уж занесение нуля в некоторую переменную перед завершением приложения является типичной ситуацией. При включении в текст модуля этого оператора я собирался написать, что диалоговая процедура снята с

выполнения и уже нет смысла что-то записывать в DlgWin, так как при следующем вызове диалога функция CreateDialog заново занесет в DlgWin дескриптор созданного окна. Но если удалить комментарий, то можно увидеть, что реакция программы на щелчок правой кнопкой по окну немодального диалога изменяется. Оказывается, запись нуля перед закрытием диалога помогла выяснить, что диалог вовсе не был уничтожен.

Используя показанную выше диалоговую процедуру, следует создавать по щелчку левой кнопкой немодальный диалог, а по щелчку правой кнопкой — модальный диалог. Модальный диалог блокирует поступление в основное окно приложения сообщений от внешних устройств, но основная функция главного окна продолжает просматривать очередь сообщений. Поэтому в основном окне можно организовать взаимодействие с модальным диалогом. Это позволит, например, закрывать диалоговое окно, если пользователь какое-то время не работает с ним. Для этого перед созданием модального диалога при помощи функции SetTimer() запускается таймер. Через заданный интервал времени в диалоговое окно посылается сообщение, вызывающее завершение диалога. В нашем примере можно послать сообщение WM_CLOSE или WM_COMMAND с идентификатором IDOK.

В таком решении есть один недостаток. Заданный интервал лучше отсчитывать не от момента создания диалога, а от последнего действия пользователя с элементами окна. Решить эту задачу можно при помощи перезапуска таймера по сообщению WM_ENTERIDLE, которое модальный диалог посылает породившему его окну при каждом выходе из диалоговой процедуры. Назначение этого сообщения уже рассматривалось ранее при обсуждении работы с меню.

Немодальный диалог сообщение WM_ENTERIDLE не посылает, поэтому нужно принудительно снимать окно диалога через семь секунд после создания.

Код модуля с оконной функцией главного окна, реализующей предложенное решение (файл WndProc.cpp), приведен в листинге 7.5.

Листинг 7.5

```

#include <windows.h>
#include "resource.h"
extern HWND DlgMod;
extern HWND DlgWin;
BOOL Dialog (HWND hDlg,UINT,WPARAM,LPARAM); //Прототип диалоговой процедуры:
LRESULT APIENTRY InputWndProc (HWND hWnd, UINT message, WPARAM wParam,
                               LPARAM lParam)
{
    switch (message)
    {
        case WM_DESTROY: PostQuitMessage(0); break;
        //По щелчку правой кнопкой мыши запускается таймер и создается
        //модальный диалог
        case WM_RBUTTONDOWN: //Создание модального диалога
            SetTimer(hWnd,1,5000,NULL);
            DialogBoxParam(GetModuleHandle(NULL),
                MAKEINTRESOURCE(DIALOGMIN),hWnd,(DLGPROC) DialogMin,17);
            DlgMod = 0;
            break;
        case WM_LBUTTONDOWN:
            SetTimer(hWnd,2,7000,NULL); //Через 7 секунд окно диалога исчезнет
            DlgWin = CreateDialog(GetModuleHandle(NULL),
                MAKEINTRESOURCE(DIALOGMIN),hWnd,(DLGPROC) DialogMin);
            break;
        //После каждой операции в модальном окне оконная процедура главного окна
        //получает сообщение WM_ENTERIDLE и таймер перезапускается
        case WM_ENTERIDLE:
            KillTimer( hWnd, 1 ); //Перезапуск
            SetTimer(hWnd,1,5000,NULL); таймера.
            break;
        //Через пять секунд после прихода последнего сообщения WM_ENTERIDLE
        //в диалоговое окно DlgMod (если оно создавалось) будет послано сообщение
        //WM_CLOSE и диалог завершится.
        case WM_TIMER:
            KillTimer( hWnd, 1 ); //Завершение модального диалога
            if(DlgMod) PostMessage(DlgMod,WM_CLOSE,0,0);
            KillTimer( hWnd, 2 ); //Завершение немодального диалога
            if(DlgWin) PostMessage(DlgWin,WM_CLOSE,0,0);
            break;
    }
}
return DefWindowProc(hWnd,message,wParam,lParam);
}

```

Легко убедиться в том, что при щелчках по окну как модального, так и немодального диалога в диалоговую процедуру поступают сообщения мыши (WM_LBUTTONDOWN, WM_RBUTTONDOWN). Но если модальное окно находится в фокусе ввода, то сообщения ему поступают из собственного цикла опроса очереди, а если фокус установлен на немодальное окно, то сообщения выбираются из очереди функцией основного цикла приложения GetMessage(). Функция DispatchMessage() основного цикла приложения пересылает сообщения в диалоговую процедуру, предварительно убедившись в том, что полученный в msg.hwnd дескриптор окна совпадает с DlgWin. Но если использовать основной цикл из предыдущих проектов, то нажатие клавиши Enter не приводит к закрытию диалога, так как не срабатывает назначение по умолчанию кнопки ОК.

Сообщения, необходимые для корректного обслуживания назначенной по умолчанию кнопки, будут посылаться в процедуру немодального диалога, если в основной цикл включить вызов функции IsDialogMessage(DlgWin, &msg), которая сама передает все сообщения, адресованные немодальному диалогу DlgWin и его элементам управления. Данная функция также определяет, что извлеченное из очереди сообщение WM_CHAR порождено нажатием клавиши Enter. Если в этот момент фокус ввода принадлежит кнопке в диалоговом окне, то функция подменяет WM_CHAR сообщением WM_COMMAND, извещающим о щелчке по кнопке. Эту же функцию необходимо использовать, чтобы немодальный диалог корректно работал с другими клавишами.

Главная функция приложения, содержащего немодальный диалог с назначенной по умолчанию кнопкой, реализована в файле Main.cpp. С учетом данного замечания она будет выглядеть так, как показано в листинге 7.6.

Листинг 7.6

```

#include <windows.h>
extern HWND DlgWin;
extern HWND DlgMod;
//Функция инициализации совпадает с проектом minim, поэтому здесь
//указывается только ее прототип
int InitApp(HINSTANCE KodPril);
HWND hWnd;
MSG msg;
int APIENTRY WinMain (HINSTANCE KodPril,HINSTANCE,LPSTR,int)
{
if (InitApp(KodPril)) return 1;
while (GetMessage(&msg, NULL, 0, 0) )
{
//Если в очереди приложения есть сообщения для диалоговых окон, то
//отображается MessageBox() – это средство задержать вызов диалоговой
//процедуры до закрытия диалога
if(msg.hwnd==DlgWin || msg.hwnd==DlgMod )
if(msg.message==WM_RBUTTONDOWN)
MessageBox(NULL,"Щелчок в окне диалога","Приложение",NULL);
if(!IsDialogMessage(DlgWin, &msg))
//Если функция IsDialogMessage() возвращает ненулевое значение, значит, она
//организовала вызов диалоговой процедуры DlgWin.
DispatchMessage( (LPMSG) &msg );
//Вызов DispatchMessage() при этом нужно пропустить, иначе диалоговое окно
//DlgWin будет получать все сообщения дважды
}
return 0;
}

```

Обратите внимание на то, что обычные сообщения функция DispatchMessage() передает диалоговому окну не хуже, чем IsDialogMessage(), и, только когда приходят сообщения от клавиатуры DispatchMessage(), обрабатывает их некорректно. Поскольку фокус ввода принадлежит не диалогу, а его дочернему окну (кнопке IDOK), в этих сообщениях msg.hwnd содержит дескриптор кнопки и DispatchMessage() не вызывает диалоговую процедуру.

Корректную обработку нажатия клавиши Enter можно организовать и без использования функции IsDialogMessage(). Для этого следует использовать дескриптор окна IDOK, который был сохранен в переменной HWND ButtonWnd. Также потребуется подменять нажатия и отпускания клавиши Enter, поступающие в окно кнопки, сообщениями WM_LBUTTONDOWN и WM_LBUTTONUP, как показано в листинге 7.7.

Листинг 7.7

```

while (GetMessage(&msg, NULL, 0, 0))
{
extern HWND ButtonWnd;
if(msg.hwnd==ButtonWnd && LOWORD(msg.wParam)==0xd &&
msg.message==WM_KEYDOWN)
{
msg.message = WM_LBUTTONDOWN;
msg.wParam = MK_LBUTTON;
msg.lParam = 0x00050005;
}
if(msg.hwnd==ButtonWnd && LOWORD(msg.wParam)==0xd &&
msg.message==WM_KEYUP)
{
msg.message = WM_LBUTTONUP;
msg.wParam = MK_LBUTTON;
msg.lParam = 0x00050005;
}
DispatchMessage((LPMSG) &msg);
}

```

Показанный выше цикл опроса очереди работает так же корректно, как и при использовании функции IsDialogMessage(), но функция IsDialogMessage() обслуживает различные клавиши и все элементы управления, размещенные в диалоговом окне, посылая и обрабатывая сообщения WM_GETDLGCODE.

Теперь следует провести анализ работы проекта.

Прежде всего следует активировать модальный диалог и при помощи щелчка правой кнопкой по рабочей области его окна отобразить сообщение о типе диалога, как показано на рис. 7.2 слева. Такой же щелчок по немодальному окну выведет сообщение не из диалогового окна, а из

основного цикла приложения, показанное на рис. 7.2 справа. Это подтверждает, что немодальный диалог получает сообщения из основного цикла приложения, а в модальном диалоге организован отдельный цикл опроса очереди.

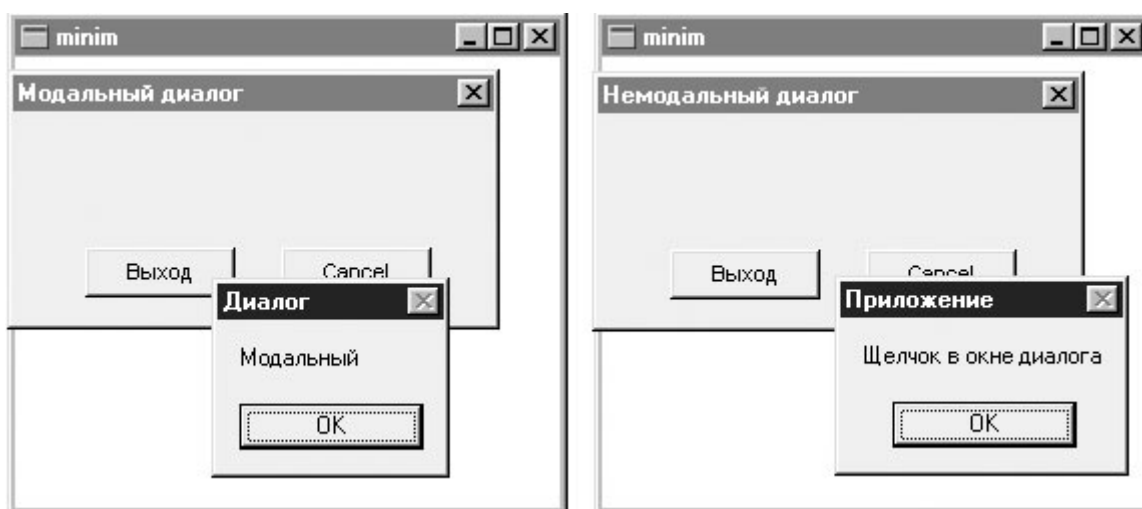


Рис. 7.2. Виды диалоговых окон

Если в ситуации, показанной на рис. 7.2 справа, не нажимать кнопку ОК в окне MessageBox, то по истечении семи секунд диалоговое окно исчезнет. Оно будет закрыто по таймеру, а в основном цикле будет находиться извлеченное из очереди, но не обработанное сообщение, предназначенное уже закрытому окну диалога.

После исчезновения окна немодального диалога, вызванного истечением времени, нужно закрыть окно с надписью Щелчок в окне диалога. Управление перейдет к функции DispatchMessage(), которая находит пусковую точку диалоговой процедуры, передает ей сообщение WM_RBUTTONDOWN, и та при закрытом окне диалога выводит сообщение Немодальный. Если после закрытия немодального диалога функцией EndDialog() послать ему, например, по таймеру сообщение, приведенное ниже, то его оконная функция будет каждый раз выводить сообщение о типе диалога:

```
if (DlgWin) PostMessage (DlgWin, WM_RBUTTONDOWN, MK_RBUTTON, 0x50005);
```

Причина подобного поведения заключается в том, что функция EndDialog() уничтожает модальный диалог. Окно немодального диалога она только закрывает, но не уничтожает. Его легко заново показать, вызвав функцию ShowWindow(DlgWin, SW_NORMAL).

Если использовать одну оконную процедуру и для модального, и для немодального диалога, то при закрытии окна необходимо учитывать его тип, используя, например, следующую конструкцию:

```
if (hDlg==DlgWin) DestroyWindow(hDlg);
else EndDialog(hDlg, TRUE);
```

Теперь нужно снова запустить проект и вызвать на экран сначала немодальный, а затем модальный диалог (рис. 7.3). Легко убедиться в том, что модальный диалог блокирует перевод фокуса ввода на основное окно, но при помощи щелчка мыши можно перенести фокус ввода на немодальный диалог и обратно. В немодальный диалог, в отличие от основного окна, продолжают поступать сообщения от кнопок мыши. Но щелчок по нему правой кнопкой сразу выводит сообщение Немодальный без предварительного извещения из основного цикла Щелчок в окне диалога.

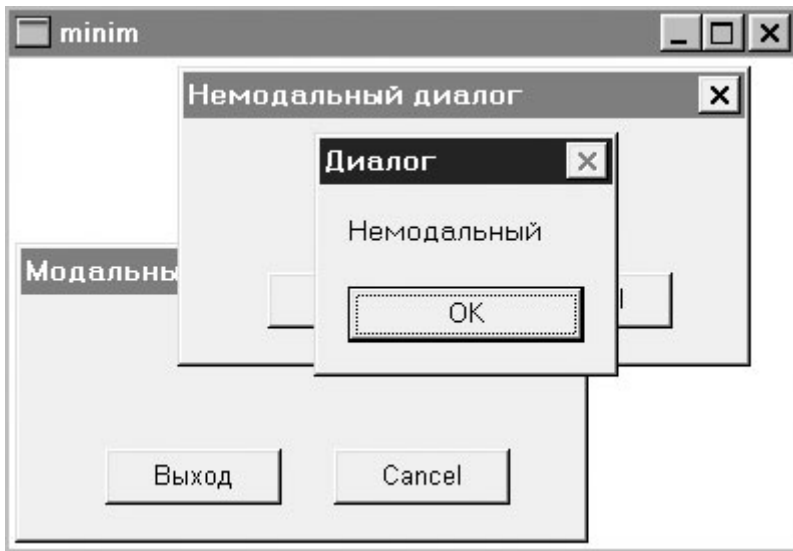


Рис. 7.3. Сообщение немодального диалога

Это указывает на то, что в данной ситуации немодальный диалог получает сообщения не через основной цикл опроса очереди приложения, а из цикла опроса очереди модального диалога. Но в модальном диалоге никто не предусмотрел функцию `IsDialogMessage()`, поэтому при одновременном выводе двух диалогов в немодальном окне перестают правильно работать клавиши `Enter` и `Tab`. Справочная система не дает рекомендаций по корректному выходу из данной ситуации.

Тот факт, что модальный диалог имеет свой цикл опроса очереди, позволяет достаточно просто организовать оконное приложение. Нужно создать диалоговое окно при помощи редактора ресурсов. При этом в текущем примере можно указать имя ресурса «`START`» и разместить в нем единственную кнопку завершения работы `IDCANCEL`. При этом нужно не забыть взвести в свойствах диалога на вкладке `More Style` флажок `Visible` и выбрать любой стиль окна, кроме `Child`.

В главной функции потребуется вызвать модальный диалог, как показано ниже:

```
int APIENTRY WinMain (HINSTANCE KodPril, HINSTANCE, LPSTR, int )
{
    DialogBox = KodPril, "START", NULL, (DLGPROC)StartProc);
    return 0;
}
```

После этого на экран будет выведено диалоговое окно, события в котором можно обрабатывать оконной процедурой диалога, код которой приведен в листинге 7.8.

Листинг 7.8

```
BOOL StartProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {
        case WM_INITDIALOG: //Здесь выполняется инициализация приложения
            return TRUE;
        case WM_LBUTTONDOWN: break;
        case WM_COMMAND:
            if (LOWORD(wParam)==IDCANCEL)
            {
                EndDialog(hDlg, LOWORD(wParam));
                return TRUE;
            }
    }
    return FALSE;
}
```

Как указывается в литературе, организация приложения на основе диалогового окна обеспечивает минимальные затраты времени процессора на работу операционной системы.