Binghamton University

EngiNet™

State University of New York

Thomas J. Watson

School of Engineering
and Applied Science

EngiNet™

WARNING

CS 560

Computer Graphics

Professor Richard Eckert

Lecture # 3

January 29, 2001

Lecture 3

Wednesday, 1-29-01

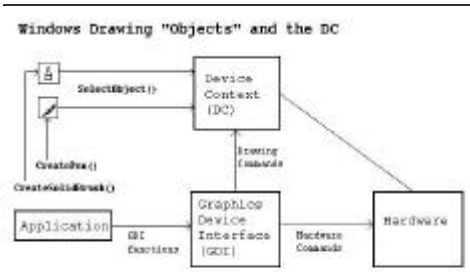
- **An Introduction to Windows Programming -- MFC**

Device Independent Graphics

- **Windows programs don't access hardware devices directly**
- **Make calls to generic drawing functions within the Windows 'Graphics Device Interface' (GDI) -- a DLL**
- **The GDI translates these into HW commands**

Device Context

- **Windows programs don't draw directly on HW**
- **Draw on "Device Context" (DC)**
  - **Abstracts the device it represents**
  - **Like a painter's canvas**
  - **Specifies drawing attribute settings**
    - **e.g., text color**
  - **Contains drawing objects**
    - **e.g., pens, brushes, bitmaps, fonts**



Colors in Windows

- Uses 4-byte numbers to represent colors
- Simplest method--direct color:
  - typedef DWORD COLORREF;

0	Blue (0-255)	Green (0-255)	Red (0-255)

  - MSB=0:
    - ==> RGB color used (default)
    - Other bytes specify R, G, B intensities

RGB() Macro

- Specify Red, Green, Blue intensities
- RGB() generates a COLORREF value
- can be used in color-setting ftns), e.g.

```
COLORREF  cr;
cr = RGB (0,0,255);   /* blue */
```

- Example usage in a program

```
SetTextColor(RGB(255,0,0)); //red text
SetBkColor(RGB(0,0,255)); //blue bkgnd
```

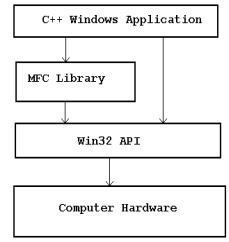
(These are member functions of CDC)

Some GDI Drawing Primitives

- Arc(x1,y1,x2,y2,xStart,yStart,xEnd,yEnd);
- Ellipse (x1,y1,x2,y2);
- MoveTo (x1,y1);
- LineTo (x1,y1);
- Polygon (points_array,nCount);
- Polyline (points_array,nCount);
- Rectangle (x1,y1,x2,y2);
- SetPixel (x1,y1,colorref);
- Many more (see on-line help on CDC)

MFC Programming

- The Microsoft Foundation Class (MFC) Library--
  - A Hierarchy of C++ classes designed to facilitate Windows programming
  - An alternative to using Win32 API functions
  - A Visual C++ Windows app can use either Win32 API MFC or both



The Relationship between Windows MFC and Win32 API Programming

Some Characteristics of MFC

- Offer convenience of REUSABLE CODE
  - Many tasks common to all Windows apps are provided by MFC
  - Programs can inherit and modify this functionality as needed
  - We don't need to recreate these tasks
  - MFC handles many clerical details in Windows programs
  - MFC Class Hierarchy (See online help on "Hierarchy Chart")

- Can lead to faster program development
  - But there's a steep learning curve--
  - Especially for newcomers to object-oriented programming
- MFC Programs must be written in C++ and require the use of classes
  - Programmer must have good grasp of
    - How classes are declared, instantiated, and used
    - Encapsulation
    - Inheritance
    - Polymorphism--virtual functions

Some important Classes

- *CObject:* "Mother of all classes"; Provides Serialization
  - *CDC:* Encapsulates the device context (Graphical Drawing)
  - *CGdiObject:* Base class for various drawing objects (bitmaps, pens, etc.)
  - *CCmdTarget:* Encapsulates message passing process & is parent of:

- *CWnd:* Base class all windows derived from:
  - *CFrameWindow:* Can contain other windows
  - *CView:* Encapsulates process of displaying data
  - *CDialog:* Encapsulates dialog boxes
- *CWinThread:* Defines a thread of execution & is parent of:
  - *CWinApp:* Encapsulates an MFC application; Controls Startup, initialization, execution, shutdown; When instantiated, application runs
- *CDocument:* Encapsulates the data associated with a program

– All MFC apps must have a CWnd-derived class and a CWinApp-derived class

Primary task in writing an MFC program

- **To create classes**
- **Most will be derived from MFC library classes**

MFC Class Member Functions

- Most functions called by an app will be members of an MFC class
- Examples:
  - *ShowWindow()*--a member of CWnd class
  - *TextOut()*--a member of CDC
  - *LoadBitmap()*--a member of CBitmap
- Applications can also call API functions directly
  - More convenient to use MFC member functions

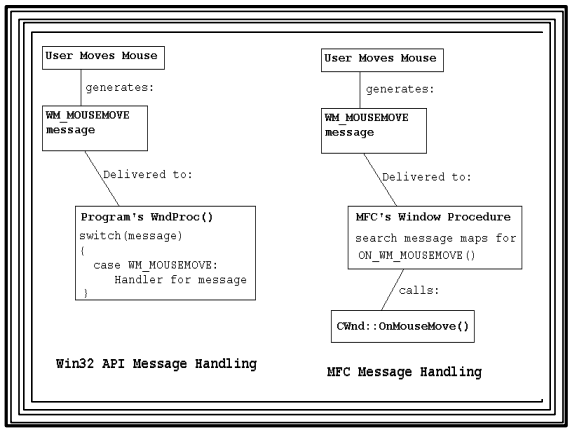
MFC Global Functions

- Not members of any MFC class
- Usually begin with Afx prefix (Application FrameworKS)
- Independent of or span MFC class hierarchy
- Example:
  - *AfxMessageBox()*--
    - Message boxes are predefined windows
    - Can be activated independently from rest of an application

Message Processing under MFC

- Like API programs, MFC programs must handle messages from Windows
- API mechanism: big switch/case statement
- MFC mechanism: "message maps" (lookup tables):
- Table entries:
  - Message number
  - Pointer to a derived class member message-processing function

- Programs must:
  - Declare message-processing functions
  - Map them to messages app is going to respond to
  - Mapping done by "message-mapping macros"
  - Most MFC app windows use a window procedure (WndProc) supplied by library
  - Message maps enable the library window procedure to find the function corresponding to the current message



MFC Windows Programming (Document/View Approach)

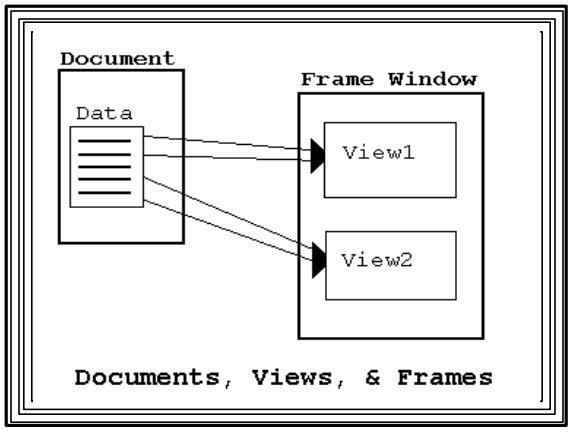
- Frequently need to have different views of same data
- Doc/View approach achieves this separation:
  - Encapsulates data in a *CDocument* class object
  - Encapsulates data display mechanism & user interaction in a *CView* class object
  - Still need to create *CFrameWnd* and *CWinApp* classes
    - But their roles are reduced

Document Interfaces

- Single Document interface (SDI) app
  - Program deals with one document at a time
- Multiple Document Interface (MDI) app
  - Program organized to handle multiple documents simultaneously
  - Multiple open documents can be of same or different types
  - Example of an MDI application: Microsoft Word

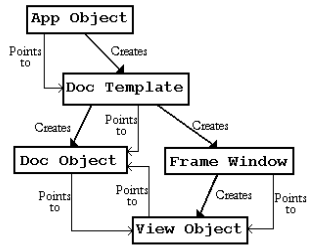
Frame Window

- Window in which a view of a document is displayed
- A document can have multiple views associated with it
  - different ways of looking at the same data
- But a view has only one document associated with it



MFC Template Class object

- Handles coordination between documents, views, and frame windows
- In general:
  - Application object creates a template:
    - which coordinates display of doc's data…
      - in a view…
        - inside a frame window



Relationship between Application, Document Template, Document, Frame Window, & View in a Document/View Approach MFC Program.

Document/View Programs

- Almost always have at least four classes derived from:
  - *CFrameWnd*
  - *Cdocument*
  - *CView*
  - *CWinApp*
- Usually put into separate declaration and implementation files
- Lots of initialization code
- Could be done by hand, but nobody does it that way

AppWizard and ClassWizard Tools

- AppWizard
  - Generates a Doc/View MFC pgm framework
    - Can be built on and customized by programmer
    - Performs required initialization automatically
      - Creates functional *CFrameWnd, CView, CDocument, CWinApp* classes
    - Gives a Full-fledged window with all common menu items, tools, etc.

ClassWizard

- Connects resources & user-generated events to program response code
- Can write skeleton routines to handle messages
- Inserts code into appropriate places in program
- Code then can then be customized by hand
- Can used to help derive classes from MFC base classes

SKETCH Application

- Example of Using AppWizard and ClassWizard
- Mouse used as a drawing pencil
- Left mouse button down
  - Line in window follows mouse motion (sketching)
- Left mouse button up
  - Sketching stops
- User clicks "Drawing Color" menu item
  - popup menu to choose drawing color
- User clicks "Clear" menu item
  - Window client area is erased

- Leave *CSketchDoc*, *CSketchApp* and *CMainFrame* classes generated by AppWizard alone
- Use ClassWizard to add sketching & color functionality to *CSketchView* class
  - App only involves user interaction

Using AppWizard to Generate SKETCH Framework

- "File | New | Projects-tab"
  - Choose "MFC AppWizard (exe)"
  - Enter name of the project (here "sketch")
  - Choose "Single document" & press "Next" button
  - In next two windows ("Step 2 of 6" and "Step 3 of 6"), press "Next" button
  - In "Step 4 of 6" window, uncheck: "Docking toolbar", "Initial status bar", "Printing and print preview"
  - Press "Finish" and "OK" buttons
- "Build" & Run application:
  - "Build | Build" & "Build | Execute" from main menu

Movie Showing how to Prepare a VC++ Document/View Project and Build It

Using ClassWizard to make it into what we want

- Sketching Requirements:
  - If left mouse button is down:
    - Each time mouse moves
      - Get current point (from mouse move message data)
      - Get a device context and pen of drawing color
      - Select pen into DC
      - Move to old point
      - Draw line from old point to current point
      - Make current point become old point
      - Select pen out of device context

Variables to be used:

- BOOLEAN m_butdn flag: left button state
- *CPoint* structures (m_ptold and m_pt): old and current points
- *CDC* object pointer (pDC): to access DC drawing functions
- COLORREF variable (m_color): current drawing color
  - (Selected by user from "Drawing Color" popup menu)

Adding variables to SketchView Class

- Select ClassView tab & expand
- Right click CSketchView & choose "Add member variable
  - Type: CPoint, Name: m_ptold, Access: Public
- Repeat for:
  - CPoint m_pt, BOOL m_butdn, COLORREF m_color, CDC* pDC

Changing the Menu

- Choose ResourceView icon in project workspace window
- Expand "sketch resources" folder and Menu subfolder
- Double click IDR_MAINFRAME
  - Menu editor appears
  - Add popup menu w/ caption "Draw Color"; Items:
    - "Red" (IDM_RED), "Green" (IDM_GREEN), "Blue" (IDM_BLUE),
    - Add another menu item called "Clear" (IDM_CLEAR)

(Uncheck "Pop-up")

Adding Code

- Use ClassWizard to add code to respond to following messages:
  - WM_LBUTTONDOWN
  - WM_LBUTTONUP
  - WM_MOUSEMOVE
  - WM_COMMAND

Code Requirements

- WM_LBUTTONDOWN:
  - Set m_butdn to TRUE & record point in m_ptold
- WM_LBUTTONUP:
  - Set m_butdn to FALSE
- WM_COMMAND
  - "Drawing Color" menu choices -- Set m_color to the appropriate COLORREF with RGB()
  - "Clear" menu choice -- Invalidate window's client area to force a WM_PAINT message

- WM_MOUSEMOVE
  - If mouse button is down (m_butdn==TRUE):
    - Get pointer to a device context, pDC, with *GetDC()*
    - Set m_pt to point data from mouse message
    - Construct solid pen of color m_color using *CPen* class constructor
    - Use *pDC*'s member functions:
      - *SelectObject()* to select pen into device context
      - *MoveTo(m_ptold)* to move it to old point
      - *LineTo(m_pt)* to draw line to new point
      - Set m_ptold equal to m_pt
      - Use *pDC*'s *SelectObject()* to select pen out of DC
      - (Automatically destructor destroys pen & DC)

Using ClassWizard to Implement Requirements

- Starting ClassWizard
  - Easiest way: Click ClassWizard Toolbar Button
    - (May or may not be visible on Developer Studio main menu bar)
    - If not displayed, to add it:
      - Right-click menu bar
      - Select "Customize..." from resulting popup
      - Under "Category:" combo box, select "View"
      - ClassWizard Toolbar Button appears
      - Drag it to the menu bar

Other ways of starting ClassWizard

- Press <Ctrl>-w or
- Select "View | ClassWizard" from Developer Studio's menu bar

Using ClassWizard to add message handlers

- Result of starting ClassWizard: "MFC ClassWizard" Dialog Box
  - "sketch" in "Project:" box, "CSketchView" in "Class name:" box
- Scroll through "Messages" list to find messages we want to respond to:
  - WM_LBUTTONDOWN
  - WM_LBUTTONUP
  - WM_MOUSEMOVE
  - WM_COMMAND

WM_LBUTTONDOWN Handler

- Press "Add Function" button
- Class Wizard adds skeleton code in right places to set up *CSketchView*'s message map
  - will contain *OnLButtonDown()* handler
- Click the "Edit Code" button
  - Taken to exact place in code where additions must be made
    - Add following after "//TODO..." line:

      m_butdn = TRUE;

      m_ptold = point;

*U*se ClassWizard in Same Way to Add WM_LBUTTONUP Handler

- Add following line of code after resulting "// TODO...":

  m_butdn = FALSE;

Use ClassWizard to add WM_MOUSEMOVE handler code:

- *if (m_butdn)*
  - *{pDC = GetDC();*
  - *m_pt = point;*
  - *CPen pen(PS_SOLID, 1, m_color);*
  - *CPen *pPenOld = pDC->SelectObject(&pen);*
  - *pDCMoveTo(m_ptold);*
  - *pDCLineTo (m_pt);*
  - *m_ptold = m_pt;*
  - *pDC->SelectObject(pPenOld);}*

Adding the WM_COMMAND menu item handlers

- Invoke ClassWizard and scroll "ObjectIDs" list (not "Messages") to "IDM_BLUE"
- Select it and choose "COMMAND" in "Messages" list box
- Click "Add Function" button
- Click "OK" in resulting "Add Member Function" dialog box

- Result:
  - ClassWizard generates *On_Blue()* handler to message map
  - Press "Edit Code" to go to skeleton handler and add following code after // TODO...

    m_Color = RGB(0,0,255);
- Do same to add *OnGreen()* and *OnRed()* handlers
- Add *OnClear()* handler in same way
- Code to be added:

  Invalidate();

Member Variable Initialization

- All data members added to *CSketchView* class must be initialized
  - If not, they will contain garbage when application begins
- From Project Workspace window choose ClassView icon
  - Expand "sketch classes" icon and *CSketchView* Class icon

– Double click on *CSketchView()* constructor
– Add following initialization code under "// TODO..." comment:

```
m_pt = m_ptold = CPoint(0,0);
m_butdn = FALSE;
m_color = RGB(0,0,0);  // initial draw color black
```

Build project

- "Build" from main menu
- If no errors, you get a functioning Document/View color sketching application