Course Schedule (by weeks)

- Introduction/Applications, Introduction to Visual C++ and Windows Programming
- Computer Graphics Hardware and Software
- Graphics Output Primitives: Scan converting lines, polygons, circles, curves, text; Introduction to OpenGL
- Display Attributes and Area Fill Algorithms
- 2-Dimensional Geometric Transformations
- 2-D Windows, Viewports, and Clipping

*** Term Examination # 1 ***

Course Schedule (by weeks)

- Interactive 2-D Graphics: Input Devices, GUI Techniques
- Segmentation, Hierarchical Modeling; PHIGS, OpenGL
- Curved lines and surfaces, parametric equations, Bezier and B-spline curves
- Animation, Sprites, Game Development, DirectX
- 3-D Graphics: Modeling & Transformations
- 3-D Graphics: Viewing and Projections

Course Schedule (by weeks)

- Hidden Surface Removal

  *** Term Examination # 2 ***
- Illumination, Reflection, Shading, Texturing, Ray Tracing, Radiosity
- Fractals, Iterated Function Systems, L-Systems, Particle Systems, Escape-time algorithms, Chaos

Introduction to Computer Graphics

Computer Graphics

- Using a computer to generate visual images
- Definition of Computer Graphics:
  - Creation, storage, manipulation, and display of models of scenes using a computer
- Interactive Computer Graphics:
  - User dynamically controls displayed image attributes by means of interactive input devices

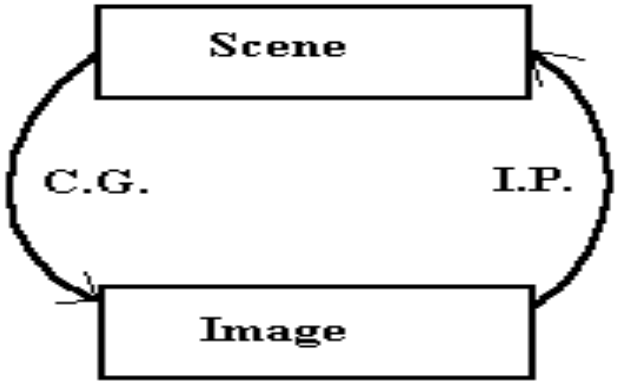
Motivation

- Human visual channel highly developed
- Efficient for communicating complex ideas

Related Field: Image Processing

- Image enhancement/understanding
- Reconstruction of objects from images
- Computer Graphics--Synthesis of images
- Image Processing--Analysis of images
- Image Processing subfields:
  - image enhancement
  - Image understanding
  - computer vision
  - pattern recognition (A.I. important)

Computer Graphics & Image Processing



Three Phases of Computer Graphics

- Modeling
  - Representing objects/scenes mathematically
- Rendering
  - Producing an image from a model
- Animation
  - Making an image move

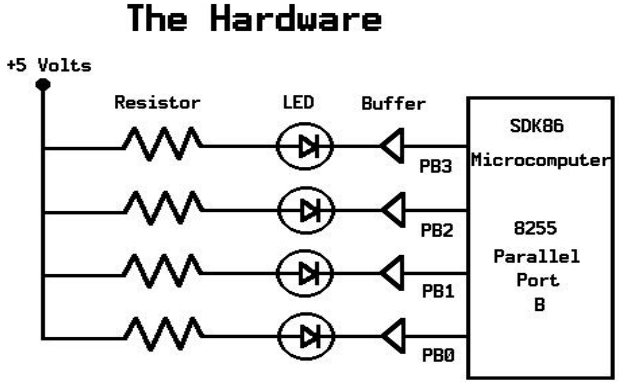
Features of Computer Graphics Models

- Output primitives:
  - building blocks
- Data structures:
  - how primitives relate to each other

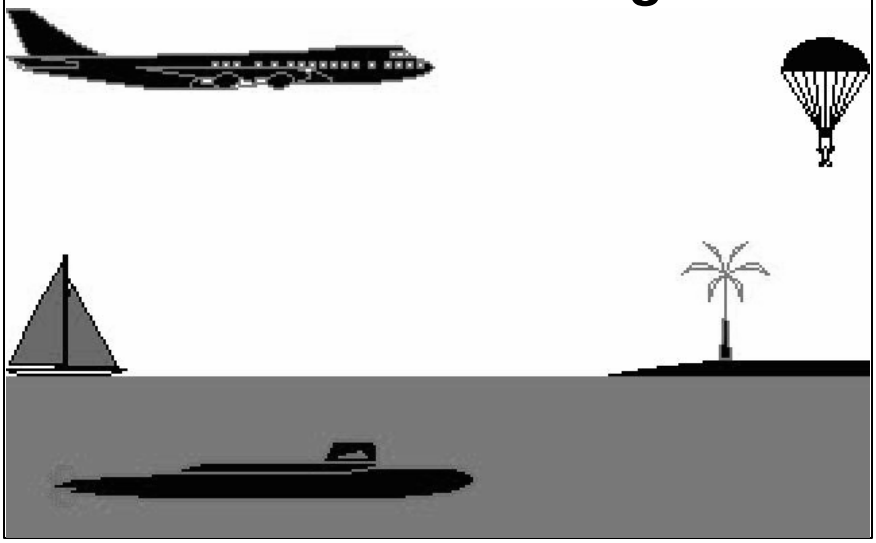
Levels of Complexity of CG

- 2-D line Drawings: Primitives
- 2-D colored images: Area fill
- 3-D line drawings: 3-D to 2-D projection
- 3-D colored images: Hidden surface removal, color, shading
- 3-D photorealistic images: materials properties, lighting, reflection, transparency, shadows (physics), complex object models
- Animation at all levels: Movement

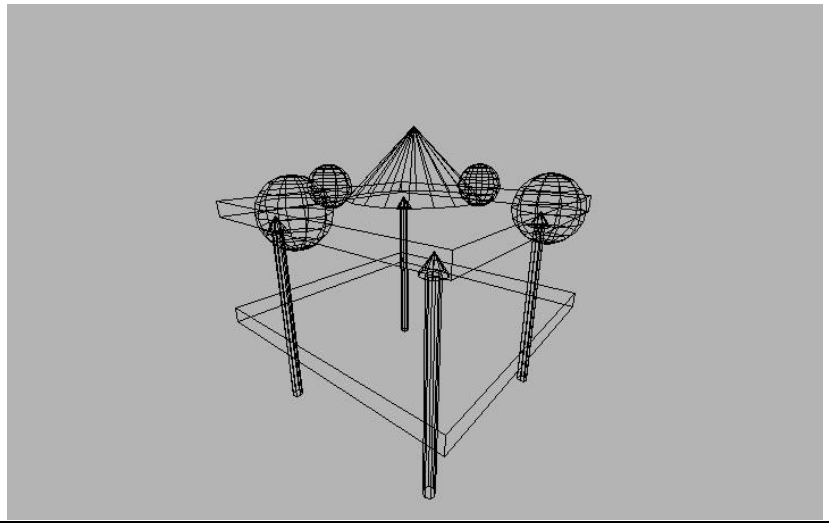
2-D Line Drawing



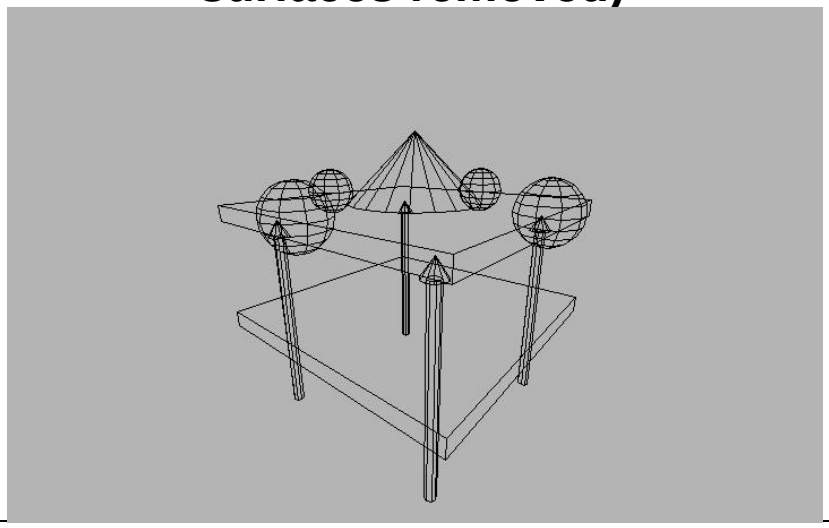
2-D Colored Image



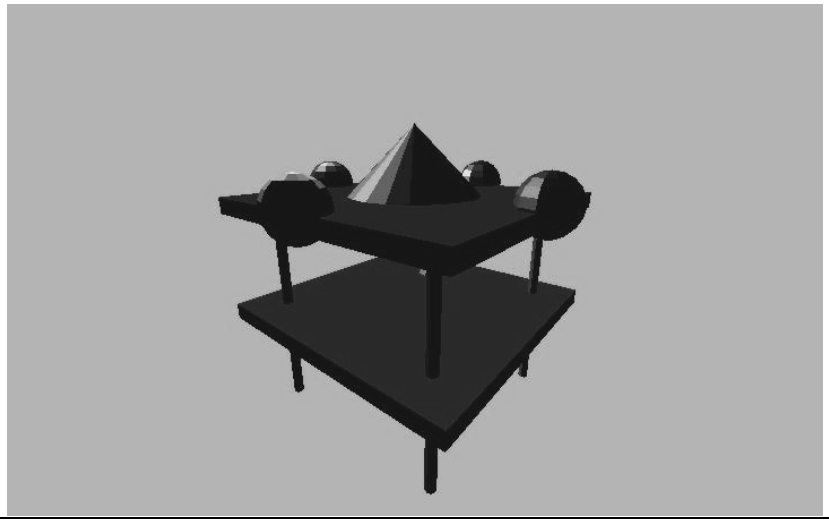
3-D Line Drawing



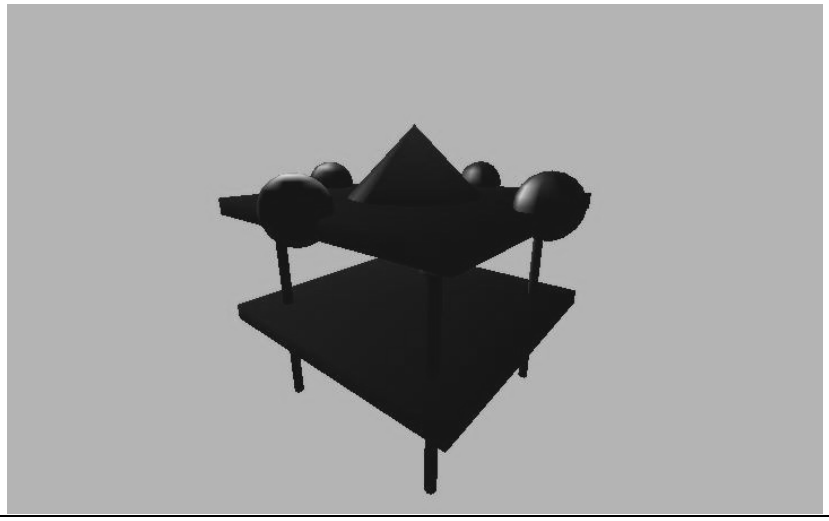
3-D Line Drawing (some hidden surfaces removed)



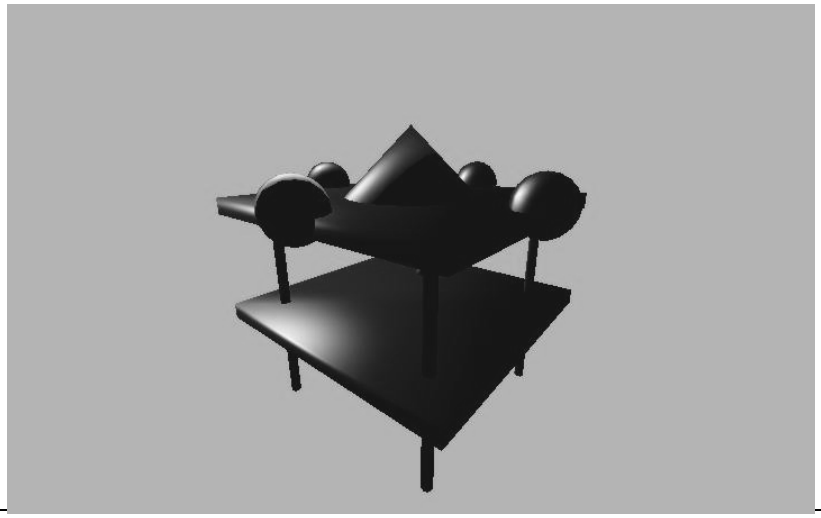
3-D Colored Image (flat shaded)



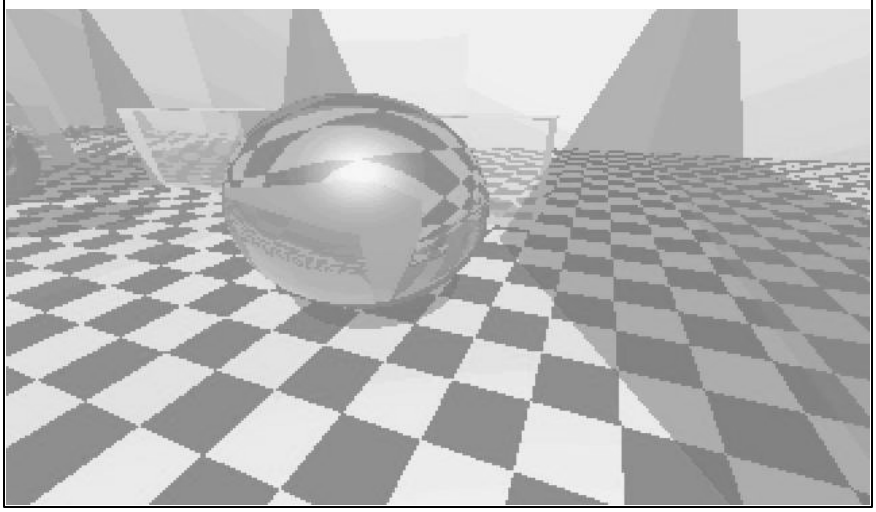
3-D Colored Image (smooth shaded)



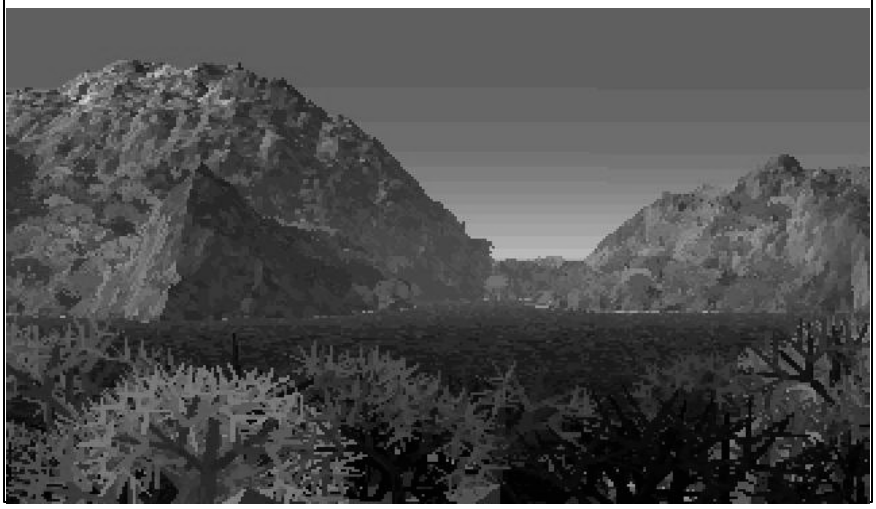
3-D Colored Image Smooth Shaded with Specular Highlights



3-D Photorealistic Image (ray traced image with texture mapping)

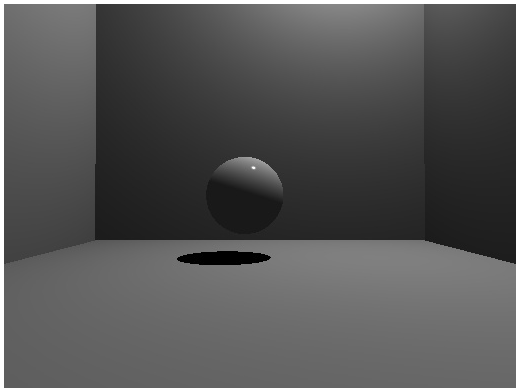


3-D Photorealistic Image (fractal mountains, L-system plants)



An Animation of a 3D Scene

- Frames generated by ray tracing

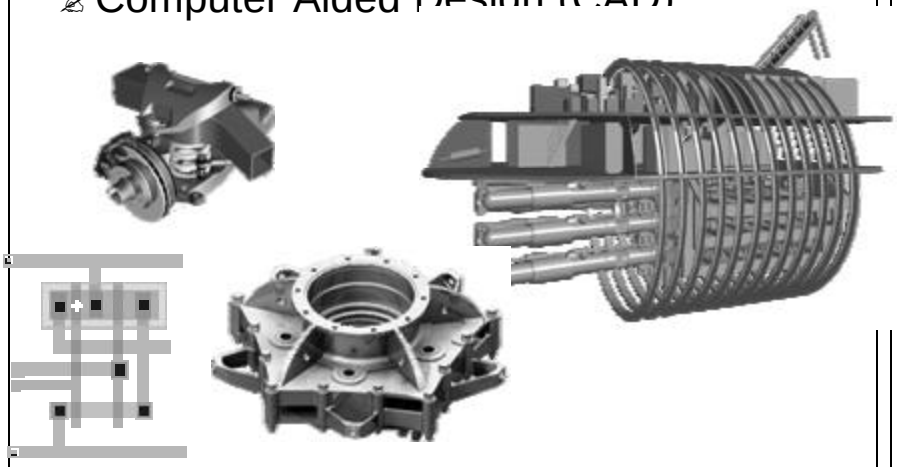


Some Applications of CG

- Data Presentation (statistics, business, scientific, demographics...)
- CAD, CAM, CIM
- Painting/Drawing systems
- TV Commercials
- Entertainment
  - Video Games
  - Motion Picture Industry
- Cartography
- Computer Art

Graphics Applications

- Computer Aided Design (CAD)



Graphics Applications

- Entertainment: Cinema

Pixar: Monster's Inc.

Video Games

- Microsoft Xbox 360
- Sony PlayStation 3
- Nintendo Wii
  - Wireless controller – Wii Remote

Video Games - Nintendo Wii

Graphics Applications

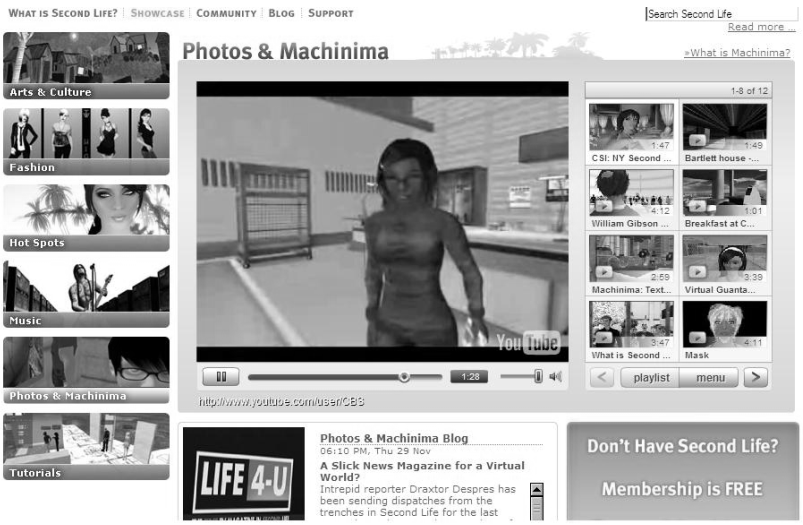
- Architectural Design
- Simulation of Reality
  - Flight simulators
  - Ground vehicle simulators
  - Arcade games
  - Virtual worlds
    - Second Life

Simulation

Driving Simulation
(Evans & Sutherland)

Flight Simulation
(NASA)

Virtual Worlds – Second Life

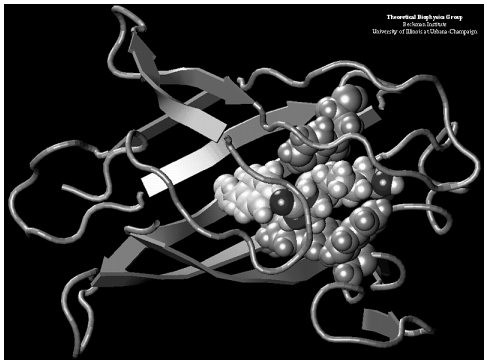


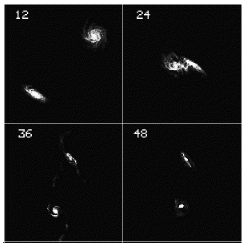
Graphics Applications

- Scientific Simulation/Visualization
  - Use graphics to make sense of vast amounts of scientific data
  - Use when too dangerous/expensive or impossible to do real experiments
- Education and Training
- Process Control
- CASE

Graphics Applications

- Scientific Visualization



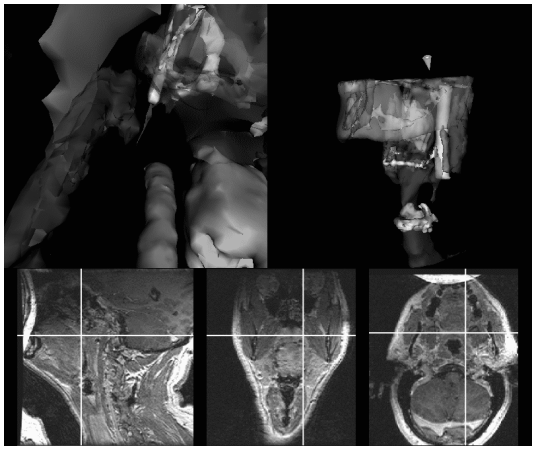


Graphics Applications

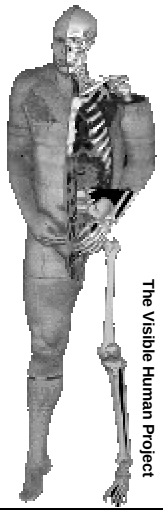
- Image Processing/Enhancement
- Medicine
  - Computed Tomography (CT Scan)
  - X-ray, ultrasound, NMR, PET:
  - All can give 3-D images of human anatomy
  - Computer-aided Surgery
- GUIs
- World Wide Web Development
- New Stuff--can't even be imagined

Graphics Applications

- Medical Visualization



MIT: Image-Guided Surgery Project



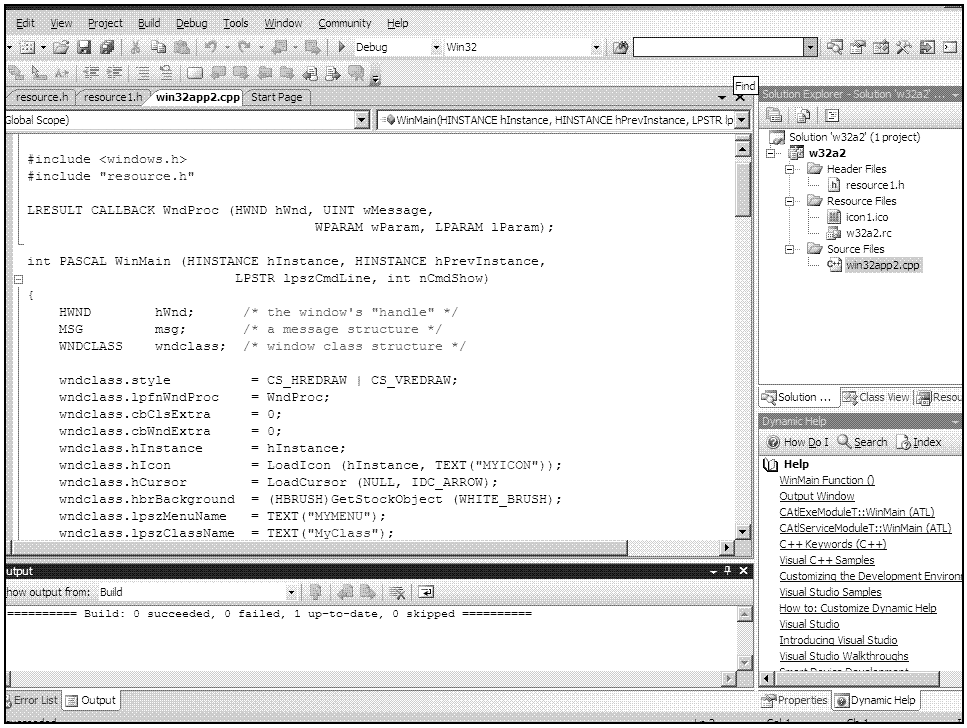
The Visible Human Project

Computer Graphics--

- A huge, fast-moving, exciting field that integrates the best of art and science
- Needs new Renaissance men & women
  - Bright and analytic enough to understand the science & math
  - Sensitive and creative enough to do the art
- Both left and right sides of the brain required!

Using Visual Studio .NET

- To prepare many kinds of applications
  - Win32 Console Applications (DOS programs)
  - Win32 API Apps in C or VC++
  - MFC Apps in VC++
  - DLLs
  - .NET Windows Forms Apps in Managed C#, VB, C++, and other languages
  - ASP.NET Web Apps and Services
  - ADO.NET Data Base Apps
  - Others including OpenGL



Solutions and Projects

- Solution
  - A single application
  - Can contain one or more projects
    - In Managed applications, projects can be in different languages
  - Overall solution information stored in a .SLN file
  - Open this when you want to work on a solution
- Project
  - Basic component of an application
  - Collection of files:
    - Source, headers, resources, settings, configuration information, many more

An Introduction to Windows Programming Using VC++

- Two approaches:
  - Win32 API
    - Most basic
  - MFC
    - Encapsulates API functions into classes
    - For most apps, easiest to use

Text and Graphics Output

- Displaying something in a window
- Text and graphics are done one pixel at a time
- Any size/shape/position possible
- Design goal: Device Independence

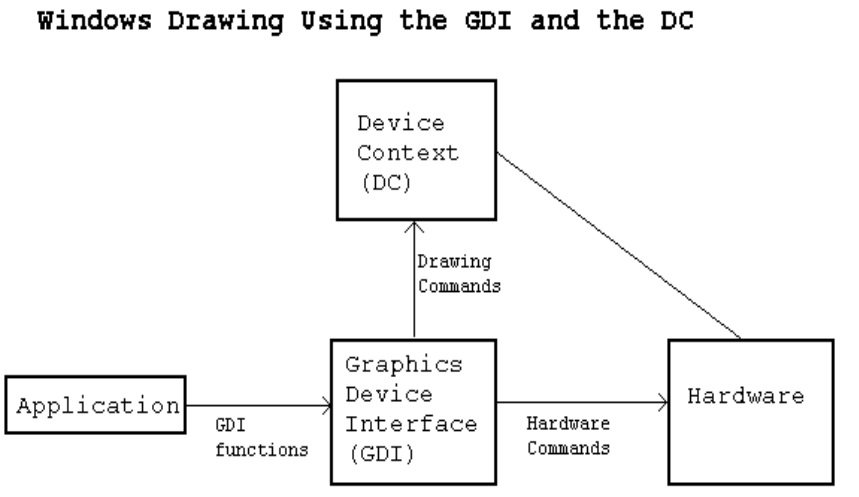
Device Independent Graphics Interface

- 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 the hardware
- Draw on "Device Context" (DC)
  - Is associated with a physical device
  - Abstracts the device it represents
  - Like a painter's canvas
  - Specifies drawing attributes
    - e.g., text color
  - Contains drawing objects
    - e.g., pens, brushes, bitmaps, fonts

The DC and the GDI



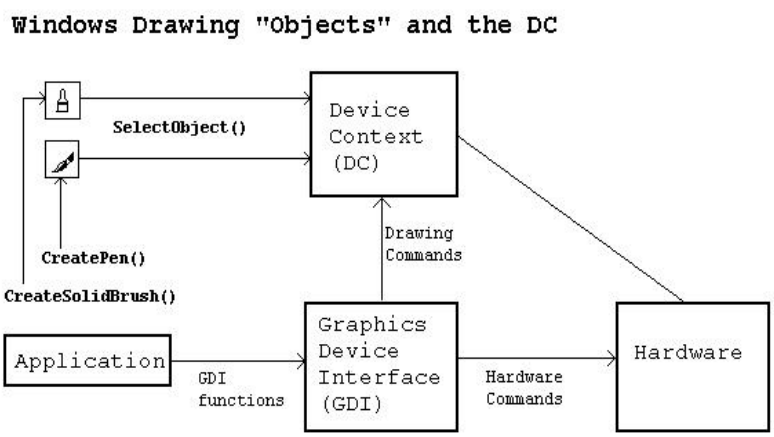
Some GDI Attributes

ATTRIBUTE	DEFAULT	FUNCTION
Background color	white	SetBkColor()
Background mode	OPAQUE	SetBkMode()
Current Position	(0,0)	MoveTo()
Drawing Mode	R2COPYPEN	SetROP2()
Mapping Mode	MM_TEXT	SetMapMode()
Text Color	Black	SetTextColor()

Some GDI Drawing Objects

Object	Default	What it is
Bitmap	none	image object
Brush	WHITE_BRUSH	area fill object
Font	SYSTEM_FONT	text font object
Pen	BLACK_PEN	line-drawing object
Color Palette	DEFAULT_PALETTE	color combinations

- Can be created with GDI functions
- Must be “selected” into a DC to be used



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 direct 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
```

A Typical Sequence With Drawing Objects:

```
HPEN    hOldP, hNewP;
HDC     hDC;
hDC = GetDC(hWnd);
hNewP = CreatePen(PS_SOLID, 3, RGB(0,0,255));
hOldP = (HPEN)SelectObject(hDC, hNewP);
// NOW DO SOME DRAWING WITH THE NEW PEN
SelectObject(hDC,hOldP); //displace pen from DC
DeleteObject(hNewP); //now can be deleted
ReleaseDC(hWnd,hDC);
```

Some GDI Drawing Primitives

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

An Example Win32 API Program

- Has Menu items to:
  - Draw a circle
  - Quit
- Types an “L” at cursor position when user left clicks the mouse
- Has an icon
- On CS-460 Sample Programs web page
  http://www.cs.binghamton.edu/~reckert/460/api.html

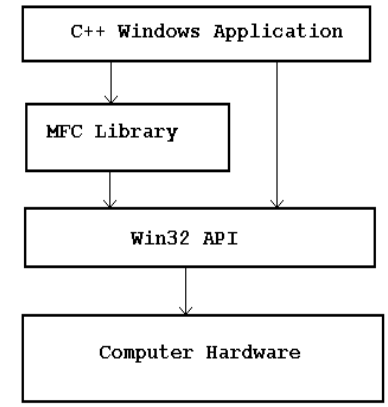
Windows Programming with MFC

MFC Programming

- MFC: The Microsoft Foundation Class Library
- Additional Notes:
  http://www.cs.binghamton.edu/~reckert/360/class14.htm
  http://www.cs.binghamton.edu/~reckert/360/class15.htm
  http://www.cs.binghamton.edu/~reckert/360/10.html

MFC

- 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

- Offers convenience of REUSABLE CODE
  - Many tasks in Windows apps are provided by MFC
  - Programs can inherit and modify this functionality as needed
  - MFC handles many clerical details in Windows pgms
  - Functionality encapsulated in MFC Classes
- Produce smaller executables
- Can lead to faster program development
- MFC Programs must be written in C++ and require the use of classes
  - Programmer must have good grasp of OO concepts

Help on MFC Classes

- See Online Help (Index) on:

  "MFC"

  "Hierarchy"

  "Hierarchy Chart"

  "MFC Reference"
- On the Web:

  http://msdn.microsoft.com/en-us/library/d06h2x6e(VS.80).aspx

Base MFC Class

- ***CObject:*** At top of hierarchy ("Mother" of almost all MFC classes)
- Provides features like:
  - Serialization
  - Runtime class information
  - Diagnostic & Debugging support
  - Some important macros
- All its functionality is inherited by any classes derived from it

Some Important Derived Classes

- ***CFile***
- ***CDC***
- ***CGdiObject***
- ***CMenu***

- ***CCmdTarget:*** Encapsulates message passing process and is parent of:
  - ***CWnd***
    - Base class from which all windows are derived
    - Encapsulates many important windows functions and data members
    - Examples:
      - m_hWnd stores the window's handle
      - *Create(…)* creates a window
  - Most common subclasses:
    - ***CFrameWindow***
    - ***CView***
    - ***CDialog***

- ***CCmdTarget*** also parent of:
  - ***CWinThread:*** Defines a thread of execution and is the parent of:
    - ***CWinApp***
      - Encapsulates an MFC application
      - Controls following aspects of Windows programs:
        - Startup, initialization, execution, the message loop, shutdown
        - An application should have one CWinApp object
        - When instantiated, application begins to run
  - ***CDocument***

Primary task in writing an MFC program

- **To create/modify classes**
  - **Most will be derived from MFC library classes**
- **Call class functions to perform tasks**

MFC Class Member Functions

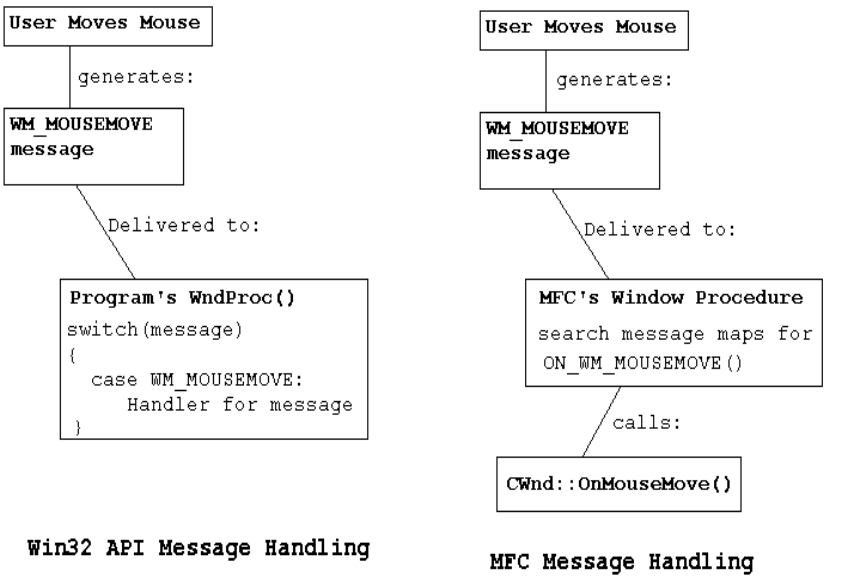
- Most functions called by an application will be members of an MFC class
- Examples:
  - *ShowWindow()***--**a member of CWnd class
  - *TextOut()***--**a member of CDC
  - *LoadBitmap()***--**a member of CBitmap
- Applications can also call API functions directly
  - Use “global scope resolution” operator ::
    - Example ::UpdateWindow(hWnd);

MFC Global Functions

- Not members of any MFC class
- Independent of or span MFC class hierarchy
- Example:
  - *AfxMessageBox()*

Message Processing under MFC

- API mechanism: switch/case statement in app's WndProc
- Under MFC, WndProc is buried in MFC framework
- Message handling mechanism: "**Message Maps**"
  - lookup tables the MFC WndProc searches
- A Message Map contains:
  - A Message number
  - A Pointer to a message-processing function
    - These are members of CWnd
    - You override the ones you want your app to respond to
    - Like virtual functions
  - "Message-mapping macros" set these up



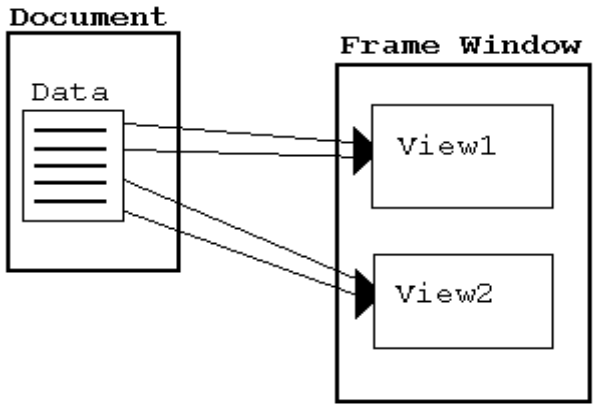
MFC Windows Programming (App/Window Approach)

- Simplest MFC programs must contain two classes derived from the hierarchy:
  - 1. An application class derived from ***CWinApp***
    - Defines the application
    - provides the message loop
  - 2. A window class usually derived from ***CWnd*** or ***CFrameWnd***
    - Defines the application's main window
- To use these & other MFC classes you must have: #include <Afxwin.h> in the .cpp file

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

Relationship between Documents, Views, and Windows



Documents, Views, & Frames

Document/View Programs

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

Microsoft Developer Studio AppWizard and ClassWizard Tools

AppWizard

- Tool that generates a Doc/View MFC program framework automatically
- Can be built on and customized by programmer
- Fast, efficient way of producing Windows Apps
- Creates functional ***CFrameWnd, CView, CDocument, CWinApp*** classes
- After AppWizard does it's thing:
  - Application can be built and run
  - Full-fledged window with all common menu items, tools, etc.

Other Visual Studio Wizards

- Dialog boxes that assist in generating code
  - Generate skeleton message handler functions
    - Set up the message map
  - Connect resources & user-generated events to program response code
  - Insert code into appropriate places in program
    - Code then can then be customized by hand
  - Create new classes or derive classes from MFC base classes
  - Add new member variables/functions to classes
- In .NET many wizards available through 'Properties window'

SKETCH Application

- Example of Using AppWizard and ClassWizard
- User can use mouse as a drawing pencil
  Left mouse button down:
  - lines in window follow mouse motion
- Left mouse button up:
  - sketching stops
- User clicks "Clear" menu item
  - window client area is erased

- Sketch data (points) won't be saved
  - So leave document (***CSketchDoc)*** class created by AppWizard alone
- Base functionality of application (***CSketchApp)*** and frame window (***CMainFrame)*** classes are adequate
  - Leave them alone
- Use ClassWizard to add sketching to CSketch***View*** class

Sketching Requirements

- Each time mouse moves:
  - If left mouse button is down:
    - Get a DC
    - Create a pen of drawing color
    - Select pen into DC
    - Move to old point
    - Draw a line to the new point
    - Make current point the old point
    - Select pen out of DC

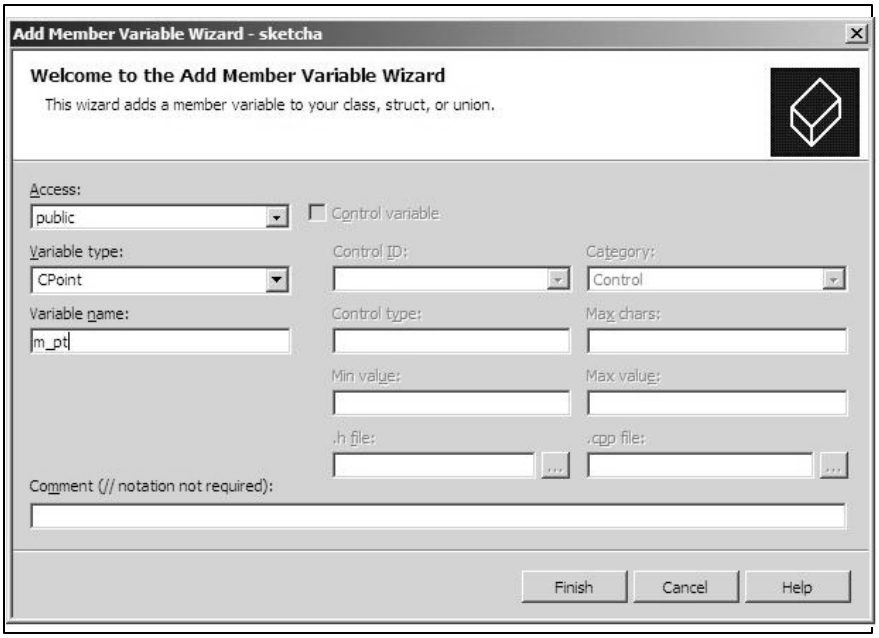
Variables

- BOOLEAN m_butdn
- CPoint m_pt, m_ptold
- COLORREF m_color
- CDC* pDC

Steps in Preparing SKETCH

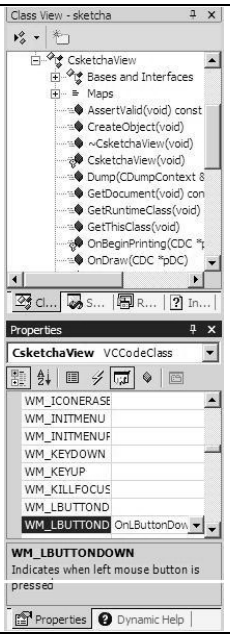
1. "File" / "New" / "Project"
   - Project Type: "Visual C++ Projects"
   - Template: "MFC Application"
   - Enter name: Sketch
2. In "Welcome to MFC Application Wizard"
   - Application type: "Single Document" Application
   - Take defaults for all other screens
3. Build Application --> Full-fledged SDI App with empty window and no functionality

4. Add member variables to CSketchView
   - Can do manually in .h file
   - Easier to:
     - Select Class View pane
     - Click on SketchView class
       - Note member functions & variables
     - Right click on CSketchView class
       - Choose "Add / Variable"
         - Launches "Add Member Variable Wizard"
       - Variable Type: enter CPoint
       - Name: m_pt
       - Access: Public (default)
         - Note after "Finish" that it's been added to the .h file
     - Repeat for other variables (or add directly in .h file):
       - CPoint m_ptold
       - bool m_butdn
       - COLORREF m_color
       - CDC* pDC



5. Add message handler functions:
   - Select CSketchView in Class View
   - Select “Messages” icon in Properties window
     - Results in a list of WM_ messages
   - Scroll to WM_LBUTTONDOWN & select it
   - Add the handler by clicking on down arrow and “<Add> OnLButtonDown”
     - Note that the function is added in the edit window and the cursor is positioned over it:
       - After “TODO…” enter following code:

```
m_butdn = TRUE;
m_ptold = point;
```



- Repeat process for WM_LBUTTONUP handler:
  - Scroll to WM_LBUTTONUP
  - Click: “<Add> OnLButtonUp”,
  - Edit Code by adding:

    m_butdn = FALSE;

- Repeat for WM_MOUSEMOVE
  - Scroll to WM_MOUSEMOVE
  - Click: “<Add> OnMouseMove”
  - Edit by adding code:

```
if (m_butdn)
{
        pDC = GetDC();
        m_pt = point;
        CPen newPen (PS_SOLID, 1, m_color);
        CPen* pPenOld = pDC->SelectObject (&newPen);
        pDC->MoveTo (m_ptold);
        pDC->LineTo (m_pt);
        m_ptold = m_pt;
        pDC->SelectObject (pPenOld);
}
```

6. Initialize variables in CSketchView constructor

- Double click on CSketchView constructor
  - CSketchView(void) in Class View
- After “TODO…”, Add code:

```
m_butdn = FALSE;
m_pt = m_ptold = CPoint(0,0);
m_color = RGB(0,0,0);
```

7. Changing Window's Properties
   - Use window's SetWindowXxxxx() functions
     - In CWinApp-derived class in its InitInstance(…) function before window is shown and updated
   - Example: Changing the default window title

```
m_pMainWnd->SetWindowTextW(
        TEXT("Sketching Application"));
```

   - There are many other SetWindowXxxxx() functions that can be used to change other properties of the window

8. Build and run the application

Menus and Command Messages

- User clicks on menu item
- WM_COMMAND message is sent
- ID_XXX identifies which menu item (its ID)
- No predefined handlers
  - We write the OnXxx() handler function
  - Must be declared in .h file and defined in .cpp file
- Event handler wizard facilitates this

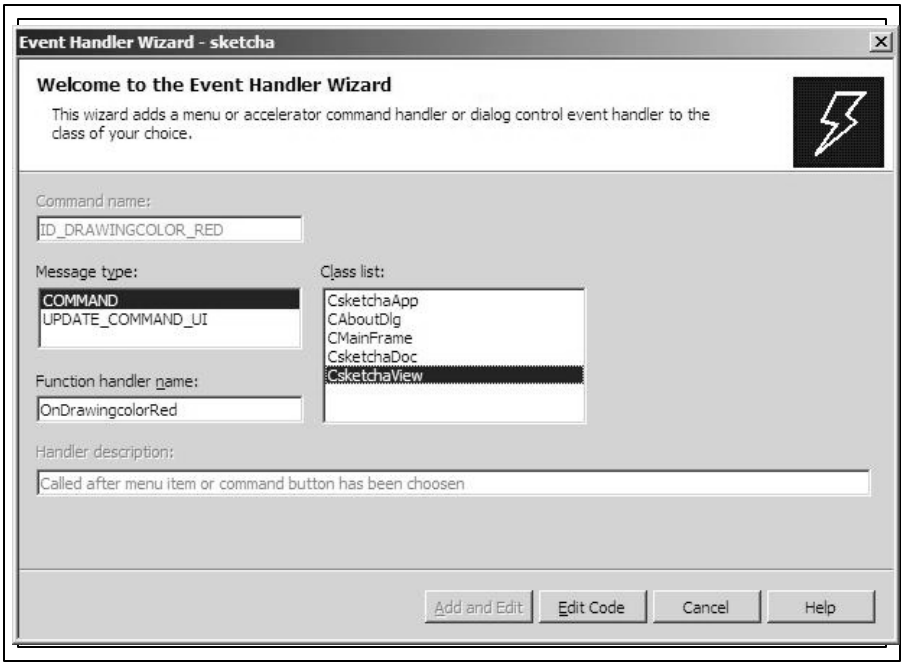
Adding Color and Clear Menu Items to SKETCH App

- Resource View (sketch.rc folder)
  - Double click Menu folder
  - Double click IDR_MAINFRAME menu
  - Add: “Drawing Color” popup menu item with items:
    - “Red”, ID_DRAWING_COLOR_RED (default ID)
    - “Blue”, ID_DRAWINGCOLOR_BLUE
    - “Green”, ID_DRAWINGCOLOR_GREEN
    - “Black”, ID_DRAWINGCOLOR_BLACK
  - Add another main menu item:
    - “Clear Screen”, ID_CLEARSCREEN
      - Set Popup property to False

Add Menu Item Command Handler Function

- One way: Use “Event Handler Wizard”
- In “Resource View” bring up menu editor
- Right click on “Red” menu item
- Select “Add Event Handler” → “Event Handler Wizard” dialog box
  - Class list: CSketchView
  - Message type: COMMAND
  - Function handler name: accept default
    - OnDrawingcolorRed
  - Click on “Add and edit”
  - After “TODO…” in editor enter following code:

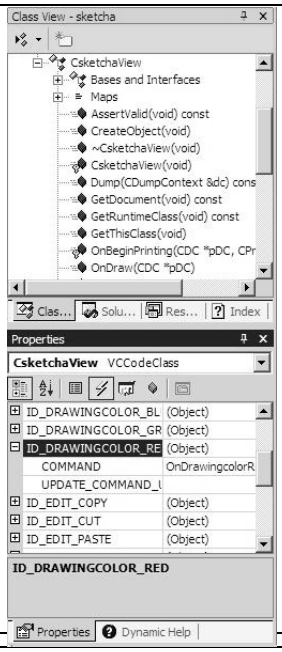
    m_color = RGB(255,0,0);



Another Method of Adding a Menu Item Command Handler

- In Class View Select CSketchView
- In Properties window select Events (lightning bolt icon)
- Scroll down to: ID_DRAWINGCOLOR_RED
- Select “COMMAND”
- Click “<Add> OnDrawingcolorRed” handler
- Edit code by adding:

  m_color = RGB(255,0,0);



Repeat for ID_DRAWINGCOLOR_BLUE
  Code: m_color = RGB(0,0,255);

Repeat for ID_DRAWINGCOLOR_GREEN
  Code: m_color = RGB(0,255,0);

Repeat for ID_DRAWINGCOLOR_BLACK
  Code: m_color = RGB(0,0,0);

Repeat for ID_CLEAR
  Code: Invalidate();

Destroying the Window

- Just need to call *DestroyWindow()*
  - Do this in the CMainFrame class – usually in response to a “Quit” menu item

Build and Run the Application