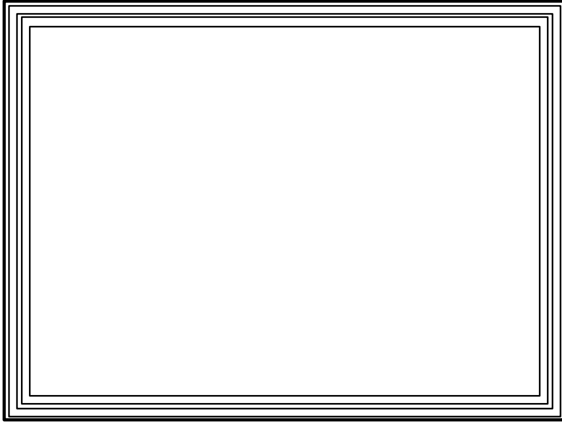


Binghamton University

EngiNet™

State University of New York

Thomas J. Watson

School of Engineering
and Applied Science

EngiNet™

WARNING

All rights reserved. No Part of this video lecture series may be reproduced in any form or by any electronic or mechanical means, including the use of information storage and retrieval systems, without written approval from the copyright owner.

©2001 The Research Foundation of the State University of New York

CS 460/560

Computer Graphics

Professor Richard Eckert

Lecture # 14

March 21, 2001

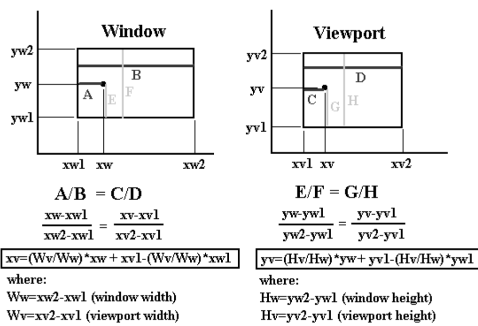
Lecture 14

A. Window to Viewport Transformation
B. Windows Mapping Modes
C. Windows DirectX
D. Clipping

2-D Viewing Transformation

- Convert from Window Coordinates to Viewport Coordinates
- (xw, yw) --> (xv,yv)
- Maps a world coordinate window to a screen coordinate viewport
- Window defined by: (xw1,yw1), (xw2,yw2)
- Viewport defined by: (xv1,yv1), (xv2,yv2)
- Basic idea is to maintain proportionality

Window to Viewport Transformation



Viewing Transformation Matrix

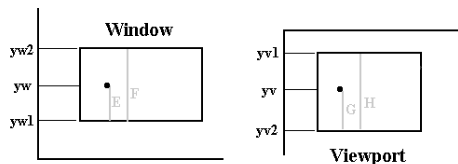
- Can cast into homogeneous matrix form:

```
        |xv|          |xw|
Pv  =   |yv|    Pw =  |yw|     Pv = Tv*Pw
        | 1|          | 1|

          | C1   0   C2 |
     Tv = | 0    C3  C4 |
          | 0    0   1  |
C1 = Wv/Ww,    C2 = xv1 - (Wv/Ww)*xw1
C3 = Hv/Hw,    C4 = yv1 - (Hv/Hw)*yw1
```

Be Careful with y-axis down

- y equation will be different!



Windows Mapping Modes

- Windows Implementation of the Viewing Transformation
- See: CS-360, CS-460/560 Notes and Sample Programs:

http://www.cs.binghamton.edu/~reckert/460/mapmodes.htm
http://www.cs.binghamton.edu/~reckert/360/class10.htm
http://www.cs.binghamton.edu/~reckert/360/mapmode1_cpp.htm
http://www.cs.binghamton.edu/~reckert/360/mapmode2_cpp.htm

Windows Mapping Modes

- Mapping Mode: a GDI Attribute
- Default DC coordinates: "Device Units"
  - For video screen: pixels
    - Origin at upper left corner of screen
  - For a printer: “dots”
    - Origin at upper left corner of printed page
- Problem--not all pixels/dots same size
  - So screen image can become a postage stamp on printer

Windows Mapping Modes

- Create logical system of units
- Windows maps output to real device
  - E.g., plot at 100,100 "logical millimeters”
  - Windows figures out where on screen
  - Not exact, but close

Windows Mapping Modes

MAPPING MODE	LOGICAL UNIT	X-AXIS	Y_AXIS
MM_TEXT	Pixel	Right	Down
MM_HIENGLISH	.001 inch	Right	Up
MM_LOENGLISH	.01 inch	Right	Up
MM_HIMETRIC	.01 mm	Right	Up
MM_LOMETRIC	.1 mm	Right	Up
MM_TWIPS	1/20 point=1/1440”	Right	Up
MM_ISOTROPIC	Arbitrary (x==y)	Selectable	
MM_ANISOTROPIC	Arbitrary (x!=y)	Selectable	

Changing the Mapping Mode

- pDC->SetMapMode(MAP_MODE);
- Maps logical to device coordinates
  - dc units: pixels, +x: right, +y: down
  - converts logical ("window") to device ("viewport") coordinates as follows

    xV = (xVExt/xWExt) * (xW - xWOrg) + xVOrg

    yV = (yVExt/yWext) * (yW - yWOrg) + yVOrg
- (xWOrg,yWOrg) and (xVOrg,yVOrg) are the origins of the window and viewport
- Both are (0,0) in the default device context

Moving Origins

- pDC->SetWindowOrg(x,y); // log. units
  - For x,y positive, think of this as moving the upper left-hand corner of the screen/paper up and right by (x,y) logical units
- pDC->SetVieportOrg(x,y); // device units--pixels
  - For x,y positive, think of this as moving the lower left-hand corner of the logical window down and right by (x,y) device units
- Both move the coordinate system origin to (x,y), but units of x,y are different

Variable Unit Mapping Modes

- Coordinate axes can have any size/orientation
- MM_ISOTROPIC-- x & y units must be same size
- MM_ANISOTROPIC -- different x and y units
- Set the X and Y scaling factors with:

  pDC->SetWindowExt (xWExt, yWExt);

  pDC->SetViewportExt (xVExt, yVExt);
- X scaling factor in going from Logical Coordinates to Device Coordinates = xVExt/xWExt
- Y scaling factor = yVExt/yWExt

Example 1

- Create coordinate system where each logical unit is two pixels:
  - twice the default device unit coordinates

```
pDC->SetMapMode (MM_ISOTROPIC);
pDC->SetWindowExt (1, 1);
pDC->SetViewportExt (2, 2);
```

Example 2

- Create coordinate system with y-axis up, each y-unit = 1/4 pixel; x-axis unchanged:

```
pDC->SetMapMode (MM_ANISOTROPIC);
pDC->SetWindowExt (1, -4);
pDC->SetViewportExt (1, 1);
```

Example 3

- Create coord system where client area is always 1000 units high & wide, y-axis up:

```
CSize size;
size = pDC->GetWindowExt (); // get client area size
  // returns size in default device units--here pixels
pDC->SetMapMode (MM_ANISOTROPIC);
pDC->SetWindowExt (1000, -1000);
pDC->SetViewportExt (size.cx, size.cy);
```

- Now (1000,1000) will always be at upper right edge of client area

DirectX and Windows Game Programming

- Game Programming
  - No "good" (fast) Windows games before 1995
  - Only DOS games for PCs--
    - Direct access to video memory permitted
    - Fast

Windows GDI

- Device independence
- Useful but slow functions
- No access to video hardware
- ==> High-speed games almost impossible

Demanding Requirements for High-resolution Animated Graphics

- 640 X 480 X 256-color ➔
  - 307,200 bytes
  - If background changes in each frame (e.g., flight simulator)
  - 307K bytes must be moved to screen
  - At least 15 times a second
  - Almost 5 Megabytes/second

- Also sprites move on a background
- Must be transferred one-by-one to screen memory too
- To avoid flicker--
  - Compose Scene in memory
  - Then transfer it to screen
    - ➔ twice as much data must be moved
  - GDI BitBlt() and StrecthBlt() can't cope with this task in real time

Microsoft Remedy (1995): the "Game SDK" (DirectX)

- A series of components called "COM objects"
  - COM: an object-oriented interface
  - Creating/using A COM interface closely resembles creating/using a C++ class

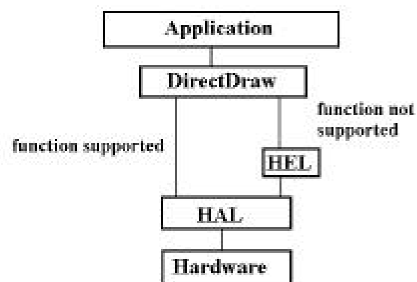
Some DirectX Components

- Direct Draw
- DirectSound
- DirectInput
- DirectPlay
- Direct3D

DirectDraw

- Provides direct control over the computer's video hardware
- Enables programs to quickly transfer graphics between memory and screen
- Takes advantage of hardware capabilities the video card
- Capabilities not available on video card are emulated in software

DirectDraw Hardware Access



DirectSound

- Provides device-independent way of directly accessing computer's sound card
- Enables programmer to add sound effects and music to games
- Can synchronize sound effects with events occurring on the screen
- Can handle 3D sound effects

DirectInput

- Provides for easy use of joystick/game controller devices
- Done in a device-independent way

DirectPlay

- Provides for implementation of multi-user games over a network or modem
- Transport-independent, protocol-independent, on-line-service-independent
- Allows Windows games to communicate with each other

Direct3D

- Provides optimized three-dimensional capabilities to Windows games
  - polygon modeling, 3D transformations, projections, clipping, surface properties, lighting, texturing, shadowing, hidden surface removal, animating
- Takes advantage of 3D acceleration hardware, if available
- No additional coding required of the game developer

DirectX games run under Windows

- Benefit from all the built-in Windows functionality
- Can use:
  - GDI graphics functions
  - All Windows user interface capabilities
  - All of fonts and other standard Windows drawing objects
  - In general, the entire Windows Win32 API

DirectDraw

- Main purpose:
  - To provide directly-accessible drawing "surfaces" in memory
  - To transfer drawing surfaces quickly to screen
- A surface:
  - A block of memory used for drawing
  - Separate surfaces used to hold each sprite in an animated scene
  - Another surface used to hold the background
  - Surfaces are composed into a final image
  - Which is transferred to the primary screen surface

Steps in Using DirectDraw in a Windows Program

- (Check the online help for details on the use of each DirectDraw function)

1. Call DirectDrawCreate () to create a DirectDraw object
2. Call the DirectDraw object's SetCooperativeLevel() member function==>
   Get control over screen and restrict access by other applications
3. Call DirectDraw object's SetDisplayMode() member function==>
   Set screen's resolution and color depth
4. Call DirectDraw object's CreateSurface() member ftn==>
   Create a primary surface object + one or more secondary surfaces (back buffers)
5. Call the primary DirectDrawSurface object's GetAttachedSurface() member function==>
   Get a pointer to a back buffer

6. Call the back buffer's DirectDrawSurface object's Lock() member function➔
   – Get a pointer to the back buffer surface's memory
7. Draw an image on the back buffer
   – **Access its memory directly**
8. Call the back buffer's DirectDrawSurface object's Unlock() member function==>
   – Tell DirectDraw that the program is done with the back buffer

9. Call the primary DirectDrawSurface object's Flip() member function➔
   - Swap surface memory associated with the primary surface and that of the next back buffer surface, thus displaying the newly-drawn image
10. When terminating the application, all direct draw objects should be removed by calling their Release() member functions

The lineminimum DirectDraw Example Application--

- Creates a 640 X 480 X 8-bit-color primary surface
- Draws 256 horizontal lines (using the current palette) on a back buffer surface
- Flips surfaces so the lines are displayed on screen
- A Win32 API program that has no menu
- Drawing action occurs in response to the user pressing the <F1> keyboard key

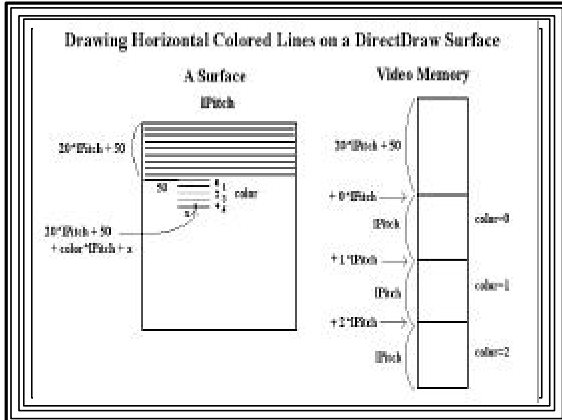
horlines.cpp

```
WinMain()
WndProc()
  VK_F1:
    Construct CDirDraw object (1-5)
    Invoke ChangeColor()
    Invoke Drawlines() (6-8)
    Call m_pPrimarySurface->flip() (9)
    Destructor gets rid of DirDraw objects (10)
```

CDirDraw Class (cdirdraw.cpp, .h)

```
Constructor()
Destructor()
Drawlines()
ChangeColor()
```

- The Program's CDirDraw class:
  – Specification in cdirdraw.h
  – Implementation in cdirdraw.cpp
    - Does most of the work in this example
    - Defines a pointer to a DirectDraw object and two pointers to DirectDraw surfaces
    - Its constructor performs steps 1 through 5 (above)
    - Its destructor performs step 10
    - Member functions ChangeColor() & DrawLines() do rest (steps 6 through 8)



- <ESC> key terminates the application
- Application keeps track of/displays time required to draw lines on back buffer
- And time required for the surface switch
  - Uses GDI TextOut() to do the display
- Program uses DirectDraw member functions in the simplest ways possible
- A robust application would do extensive error checking after most of function calls (Refer to the references.)

DirectX and lineminimum Details

- See following CS-360 Web Pages:
- Course Notes
- Class 4x--DirectX and Windows Game Programming

http://www.cs.binghamton.edu/~reckert/360/class4x.htm

- Sample Programs
- Example 4-6: lineminimum DirectX Example

http://www.cs.binghamton.edu/~reckert/360/horlines_cpp.htm
http://www.cs.binghamton.edu/~reckert/360/cdirdraw_cpp.htm
http://www.cs.binghamton.edu/~reckert/360/cdirdraw_h.htm