The Sound

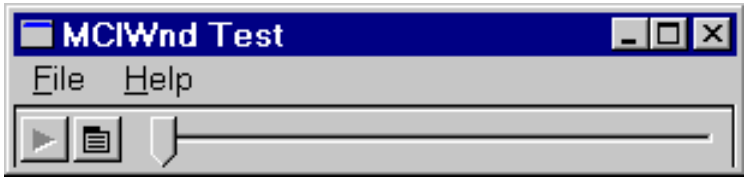
- **`MessageBeep()`**
  - a sound of a specified type, defined on the system level by the user
- MIDI (Musical Instrument Digital Interface)
  - use MCI, stream buffers or MIDI services
- WAVE
  - **`PlaySound()`** – a sound played from a file, resource or system settings
  - allows to play the sound in a loop
- MCI (Media Control Interface)
- DirectX
  - DirectSound
  - DirectMusic

Media Control Interface (MCI)

- A set of universal functions for multimedia devices (audio output, MIDI, CD, video)
- MCI device – any device which uses MCI commands
- MCI commands
  - system:
    break, sysinfo
  - obligatory for all devices:
    capability, close, info, open, status
  - basic:
    load, pause, play, record, resume, save, seek, set, status, stop
  - extended:
    configure, setaudio, setvideo, freeze, unfreeze

MCIWnd

- A window class with built-in support for playing multimedia



- Using:
  - create/destroy - `MCIWndCreate()`, `MCIWndDestroy()`
  - open/close file or device - `MCIWndOpen()`, `MCIWndClose()`
  - play - `MCIWndPlay()`, `MCIWndPause()`, `MDIWndResume()`
  - limit the play time - `MCIWndPlayFrom()`, `MCIWndPlayTo()`, `MCIWndPlayFromTo()`
  - record - `MCIWndNew()`, `MCIWndSaveDialog()`, a flag `MCIWNDF_RECORD` for `MCIWndCreate()`

The Sound in .NET

- The `System.Media` namespace [2.0]
  - `SoundPlayer`
    - only .wav files
    - `SoundLocation` – a file path or Web address
      `Stream` – an input stream
    - synchronous and asynchronous loading and playing
    - possibility to play a sound with an automatic looping
  - `SystemSounds`
    - static methods returning `SystemSound` objects for Asterisk, Beep, Exclamation, Hand, and Question sound events
  - `SystemSound`
    - the `Play()` method

```
SystemSounds.Asterisk.Play();
```

MP3 in .NET

- There is no support for MP3 in the .NET Framework
- Use DllImport attribute for MCI functions
  - ready to use, free solution from Mentalis.org: **http://www.mentalis.org/soft/class.qpx?id=1**

Windows Media Player Control

- Windows Media Player ActiveX
  - HTML in Internet browsers (IE, Firefox, Netscape Navigator)
  - Microsoft Visual C++
  - Microsoft Visual Basic 6.0
  - .NET Framework
  - Microsoft Office
- Windows Media Player Mobile ActiveX
  - HTML in Microsoft Pocket IE
  - eMbedded Visual C++
  - Active Template Library
  - can be also used from .NET Compact Framework

Windows Media Player SDK Elements

- Windows Media Player Skins
- Windows Media Player Plug-ins
  - Windows Media Player Custom Visualizations
  - Windows Media Player User Interface Plug-ins
  - Windows Media Player DSP Plug-ins
  - Windows Media Player Conversion Plug-ins
- Windows Media Metafiles
- Windows Media Player Online Stores

DirectX

- A set of multimedia technologies for Windows
- A part of Windows 98/Me/2000/XP/2003
- API (Application Programming Interface)
  - hardware control (display, sound, keyboard, mouse, joystick)
  - using modern hardware possibilities
  - hardware checking and adapting
  - high efficiency
  - HAL (Hardware Abstraction Layer) - hardware independence, using manufacturers' drivers
- DirectX SDK (Software Development Kit)
  DirectX Redistributable

Versions and Components of DirectX

- DirectX versions
  - 10 – included in Windows Vista (exclusive for this system)
  - 9 – included in Windows Server 2003
  - 8.1 – included in Windows XP
- Components:
  - DirectX Graphics
  - DirectInput
  - DirectSound, DirectMusic
- Older components:
  - DirectDraw – included in Direct3D
  - DirectPlay – deprecated (guidelines: use Windows Sockets and the Windows Firewall API)
  - DirectShow – moved from DirectX SDK to Platform SDK
  - DirectMusic – no longer supported

Compilation of DirectX Projects

- Header files [.h]
  - add **`<DXSDK>\Include`** to the list of directories with header files
  - it must be the first directory on the list
- Libraries [.lib]
  - add **`<DXSDK>\Lib\[x64,x86]`** to the list of directories with libraries added to the project
- If targeted version of DirectX is previous than current, declare preprocessor's constant
  - e.g. **`#define DIRECT3D_VERSION 0x0700`**
- Using GUID (since 8.0 version)
  - add Dxguid.lib library to the project
  - remove **`#define INITGUID`** from all source files

DirectX Graphics - Initialization

```
// create Direct3D object
IDirect3D9 *pD3D = NULL;
pD3D = Direct3DCreate9(D3D_SDK_VERSION);
// get the display mode
D3DDISPLAYMODE d3ddm;
pD3D->GetAdapterDisplayMode(D3DADAPTER_DEFAULT,
                            &d3ddm);
// create Direct3D device
IDirect3DDevice9 *pDev = NULL;
D3DPRESENT_PARAMETERS d3dpp;
ZeroMemory(&d3dpp, sizeof(d3dpp));
d3dpp.Windowed = TRUE;
d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
d3dpp.BackBufferFormat = d3ddm.Format;
g_pD3D->CreateDevice(D3DADAPTER_DEFAULT,
          D3DDEVTYPE_HAL, hWnd,
          D3DCREATE_SOFTWARE_VERTEXPROCESSING,
               &d3dpp, &pDev);
```

DirectX Graphics - drawing

```
// clear the buffer
pDev->Clear(0, NULL, D3DCLEAR_TARGET,
    D3DCOLOR_XRGB(0,0,255), 1.0f, 0);

// begin drawing the scene
pDev->BeginScene();

... // drawing

// end drawing the scene
pDev->EndScene();

// display drawn scene
pDev->Present(NULL, NULL, NULL, NULL);
```

DirectX Graphics – Releasing of Objects

- All used DirectX object must be released

```
if (pDev != NULL)
    pDev->Release();
if (pD3D != NULL)
    pD3D->Release();
```

- Sometimes it is necessary to create and destroy DirectX objects many times
  - e.g. after changing resolution or a number of colors

DirectX Graphics - Vertices

- Creating

```
CUSTOMVERTEX arV[] = { ... };
IDirect3DVertexBuffer9  *pVB;
pDev->CreateVertexBuffer(..., &pVB)

VOID* pV;
pVB->Lock(0, sizeof(arV), (BYTE**)&pV, 0);
memcpy(pV, arVertices, sizeof(arVertices));
pVB->Unlock();
```

- Drawing

```
pDev->SetStreamSource(0, pVB,
    sizeof(CUSTOMVERTEX);
pDev->SetVertexShader(D3DFVF_CUSTOMVERTEX);
pDev->DrawPrimitive(D3DPT_TRANGLELIST, 0,1);
```

DirectX Graphics - Matrices

- World transformation

```
D3DXMATRIX matWorld;
D3DXMatrixRotationY(&matWorld,
    timeGetTime()/150.0f);
pDev->SetTransform(D3DTS_WORLD, &matWorld);
```

- View transformation

```
D3DXMATRIX matView;
D3DXMatrixLookAtLH(&matView, &D3DXVECTOR3(..),
    &D3DXVECTOR3(..), &D3DXVECTOR3(..));
pDev->SetTransform(D3DTS_VIEW, &matView);
```

- Projection transformation

```
D3DXMATRIX matProj;
D3DXMatrixPerspectiveFovLH(&matProj,
    D3DX_PI/4, 1.0f, 1.0f, 100.0f );
pDev->SetTransform(D3DTS_PROJECTION,&matProj);
```

DirectX Graphics - Lights

- Each vertex must have information about its normal vector

```
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ|D3DFVF_NORMAL)
```

- The material

```
D3DMATERIAL9 mtrl;
...
pDev->SetMaterial(&mtrl);
```

- The light (ambient, point light, directional light, spotlight)

```
D3DLIGHT9 light;
...
pDev->SetLight(0, &light);
pDev->LightEnable(0, TRUE);
pDev->SetRenderState(D3DRS_LIGHTING, TRUE);
pDev->SetRenderState(D3DRS_AMBIENT,0x00202020);
```

DirectX Graphics - Textures

- Each vertex must have information about its coordinates in the texture

```
#define D3DFVF_CUSTOMVERTEX \
      (D3DFVF_XYZ|D3DFVF_DIFFUSE|D3DFVF_TEX1)
```

- Initializing

```
IDirect3DTexture8 *pTexture;
D3DXCreateTextureFromFile(pDev, "texture.bmp",
    &pTexture);
```

- Drawing

```
pDev->SetTexture(0, pTexture);
pDev->SetTextureStageState(0, D3DTSS_..., ...);
```

DirectX Graphics - Meshes

- Can be loaded from **.x** files
- Loading

```
ID3DXMesh *pMesh;
ID3DXBuffer *pBuffer;
D3DXLoadMeshFromX("tiger.x",
      D3DXMESH_SYSTEMMEM, pDev, NULL,
      &pBuffer, &dwNumMaterials, &pMesh);
// next - information about materials
```

- Drawing

```
// for each loaded subset of objects
pDev->SetMaterial(&g_pMeshMaterials[i]);
pDev->SetTexture(0, g_pMeshTextures[i]);
pMesh->DrawSubset(i);
```

DirectX Graphics - Features

- The viewport - a rectangle used to draw 3D scene
- Clipping (using coordinates)
- Fog
- Transarency
- Z-buffer
- Antialiasing – soft edges
- Bump mapping – simulates unevenness of surface
- Environment mapping – simulates image reflection
- Geometry blending – smooth joining of segments
- HLSL (high-level shader language) shaders

DirectSound

- Features:
  - playing sounds from files or resources in the WAV format
  - playing multiple sounds simultaneously
  - assigning high-priority sounds to hardware-controlled buffers
  - locating sounds in a customizable 3D environment
  - adding effects such as echo and chorus, and change effect parameters dynamically
  - capturing WAV sounds from a microphone or other input

DirectInput

- API for game control devices (mouse, keyboard joystick, force-feedback)
- Features:
  - using new devices (unknown for Win32 API)
  - faster communication with hardware
  - ignoring some settings from the control panel (it works directly with the device drivers)
  - receiving data from devices in application running in the background
  - receiving data from unknown devices

DirectInput - Scenario

1. Create the DirectInput object
2. Enumerate devices
3. Create the DirectInputDevice object for each used device
4. Specify device's settings
   - the level of cooperation
   - the data format
   - the buffer and its size
5. Get information about readiness to receive data
6. Retrieve data from the device
7. Modify device's state (if necessary)
8. Release all devices, close DirectInput

DirectSetup

- Simple API for creating DirectX components installers
- DirectXSetup() function
  - overwrites components already installed in the system, but COM guarantees backward compatibility
  - uses the system information of graphics and sound devices
  - parameters:
    - a handle to the parent's window or installer's dialog box
    - a path to DirectX files
    - flags specifying which components should be installed

DirectX for Managed Code (.NET)

- Features:
  - no COM (Component Object Model) interoperability layer – improved performance
  - more intuitive interface
  - no need to release objects

- Most DirectX samples are rewritten using C#

`Microsoft.DirectX` Namespace

- **`Microsoft.DirectX`**
  - utility operations, exception handling, matrices, quaternions, vectors, clipping plane
- **`Microsoft.DirectX.AudioVideoPlayback`**
  - basic playback and simple control of audio and video files
- **`Microsoft.DirectX.Diagnostics`**
  - using DirectX Diagnostics Tool
- **`Microsoft.DirectX.Direct3D`**
- **`Microsoft.DirectX.DirectInput`**
- **`Microsoft.DirectX.DirectSound`**

`Microsoft.DirectX` Namespace cont.

- **`Microsoft.DirectX.PrivateImplementationDetails`**
  - access to the unmanaged portions of the DirectX API
- **`Microsoft.DirectX.Security`**
  - controls permissions related to Direct3D, DirectInput, DirectPlay, and DirectSound

- Deprecated:
  - **`Microsoft.DirectX.DirectDraw`**
  - **`Microsoft.DirectX.DirectPlay`**

Managed Direct3D – Using in a Form

```
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;

class MyDirectXForm : Form{
  Device device = null;
  private bool InitializeGraphics() { ... }
  private void Render() { ... }

  static void Main() {
    using (MyDirectXForm frm = new MyDirectXForm) {
    if (!frm.InitializeGraphics()) {
      MessageBox.Show("Could not initialize Direct3D");
      return;
    }
    frm.Show();
    while (frm.Created) {
      frm.Render();
      Application.DoEvents();
    }
  }
}
```

Managed Direct3D - Initialization

```
public bool InitializeGraphics() {
  try {
    PresentParameters presentParams = new
                                PresentParameters();
    presentParams.Windowed = true;
    presentParams.SwapEffect = SwapEffect.Discard;
    // and many more parameters

    device = new Device(0, DeviceType.Hardware, this,
                  CreateFlags.SoftwareVertexProcessing,
                  presentParams);
    return true;
  }
  catch (DirectXException) {
    return false;
  }
}
```

Managed Direct 3D - Events

- Two types of events:
  - **`DeviceLost`** – e.g. when another application starts
  - **`Reset`** – e.g. changing the windowing mode
- A usual response for the event: calling the **`device.Reset()`** method
  - but first release all resources (textures, objects, buffers, etc.)

```
public void OnResetDevice(object sender,EventArgs e) {
    // ...
}
public void OnLostDevice(object sender, EventArgs e ) {
    // ...
}

device.DeviceReset +=
    new System.EventHandler(this.OnResetDevice);
device.DeviceLost +=
    new System.EventHandler(this.OnLostDevice);
```

Managed Direct3D - Drawing

```
private void Render() {
  if (device == null) {
    return;
  }

  device.Clear(ClearFlags.Target | ClearFlags.ZBuffer,
               System.Drawing.Color.Blue, 1.0f, 0);
  device.BeginScene();

  // draw the scene

  device.EndScene();
  device.Present();
}

protected override void OnPaint(PaintEventArgs e) {
  Render();
}
```

Managed Direct3D – Examples: Vertices

```
VertexElements = new VertexElement[] {
  new VertexElement(0, 0,
    DeclarationType.Float3,
    DeclarationMethod.Default,
    DeclarationUsage.Position,
    0),
  new VertexElement(0, 12,
    DeclarationType.Float3,
    DeclarationMethod.Default,
    DeclarationUsage.Normal,
    0),
  new VertexElement(0, 24,
    DeclarationType.Float2,
    DeclarationMethod.Default,
    DeclarationUsage.TextureCoordinate,
    0),
  VertexElement.VertexDeclarationEnd
};
VertexDeclarator = new VertexDeclaration(device,
VertexElements);
Vertices = new VertexBuffer(device, NumberOfVertices * 32,
  Usage.WriteOnly, VertexFormats.None, Pool.Managed);
```

Managed Direct3D – Examples: Vertices cont.

```
VertexDeclarator = new VertexDeclaration(
  device, VertexElements);
Vertices = new VertexBuffer(device,
  NumberOfVertices * 32, Usage.WriteOnly,
  VertexFormats.None, Pool.Managed);
GraphicsStream stream = null;
stream = Vertices.Lock(0, 0, LockFlags.None);
stream.Write(vertices);
Vertices.Unlock();
Indices = new IndexBuffer(device,
  NumberOfFaces * 3 * 16,
  Usage.WriteOnly, Pool.Managed, true );
Indices.SetData(faces, 0, LockFlags.None);
```

```
// drawing:
device.VertexDeclaration = VertexDeclaration;
device.SetStreamSource(0, Vertices, 0, 32);
device.Indices = Indices;
device.DrawIndexedPrimitives(
  PrimitiveType.TriangleList, 0, 0,
  NumberOfVertices, 0, NumberOfFaces);
```

Managed Direct3D – Examples: Lights

```
// turn on lights
Device.RenderState.Lighting = true;

// an optional ambient light
Device.RenderState.Ambient = System.Drawing.Color.Gray;

device.Lights[Id].Enabled = true;
device.Lights[Id].Ambient = Light.Ambient;
device.Lights[Id].Attenuation0 = Light.Attenuation0;
device.Lights[Id].Attenuation1 = Light.Attenuation1;
device.Lights[Id].Attenuation2 = Light.Attenuation2;
device.Lights[Id].Diffuse = Light.Diffuse;
device.Lights[Id].Direction = Light.Direction;
device.Lights[Id].Falloff = Light.Falloff;
device.Lights[Id].InnerConeAngle = Light.InnerConeAngle;
device.Lights[Id].OuterConeAngle = Light.OuterConeAngle;
device.Lights[Id].Position = Light.Position;
device.Lights[Id].Range = Light.Range;
device.Lights[Id].Specular = Light.Specular;
device.Lights[Id].Type = Light.Type;
```

OpenGL

- A programmable interface for graphical devices
  - it renders 2D and 3D objects discribed by a set of vertices or by pixels
- Hardware and system portability
- Windows: 95/NT
- Components:
  - core OpenGL functions
    #include <GL/gl.h>
    opengl32.lib
  - OpenGL Utility library
    #include <GL/glu.h>
    glu32.lib
  - OpenGL Programming Guide auxiliary library
    #include <GL/glaux.h>
    glaux.lib