Talk Outline

- Platform Overview
  - Introducing Metro style games
- App Model
  - Object model for app structure
  - a note about syntax
- Presentation Model
  - Screen layouts
- State and Storage
  - Persisting saved games and scores
- Metro Features
  - Make your game look professional
- Input
  - Choices of control schemes
- Audio and Media
  - Sound effects
- Networking
  - Multiplayer
Advantages of Metro style apps

- Fast and fluid UI interaction
  - “Touch first”
- Continuous app operation
- App autonomy/independence
- App collaboration
- New hardware form factors
- Reduced power consumption
- Clear development process
  - Tools, docs, APIs
- Efficient distribution
Why target Windows 8?

- Why target this platform as a game developer?
  - Easy access by developers
  - Easy development because of simplified APIs and tools
  - Large customer base
    - Only way to target new ARM-based devices
  - New opportunity
Very Short Demo

- This is why Windows 8 is great for games
  - Apps can go full screen
    - No OS chrome to clash with visuals
  - Application centric
    - No OS UI is using ANY of your pixels
  - Simplified UI for a broader audience of users
  - Fast/fluid
    - High-performance interaction is great for games
  - Multi-application environment
    - Enables casual games to appear next to another app
    - Or social apps to appear next to your game
Metro Style App Development Portal

  - Set Tag to “DirectX”
  - Push [Filter]
  - Scroll down
Platform Architecture

Metro style apps

WinRT APIs

Communication & Data

Graphics & Media

Devices & Printing

Application Model

Desktop apps

Windows Kernel Services

View

Model

Controller

System Services

Kernel
Metro style app APIs

User Interface
- HTML5/CSS
- XAML
- DirectX
- Controls
- Data Binding
- SVG
- Tiles
- Input
- Accessibility
- Printing

Devices
- Geolocation
- Portable
- Sensors
- NFC

Communications & Data
- Contracts
- Local & Cloud Storage
- Web
- Notifications
- Streams
- SMS
- Background Transfer
- XML
- Networking

Media
- Playback
- Capture
- PlayTo
- Visual Effects

Fundamentals
- Application Services
- Threading/Timers
- Memory Management
- Authentication
- Cryptography
- Globalization
Language/Component Combinations

- XAML can be used with C++ or C# apps
- DirectX can be used with C++ (or C# via wrappers)

- JavaScript uses HTML for GUI/Markup and Canvas for rendering
  - Uses core engine from IE
- Most Windows features are exposed in JavaScript in a WWA
  - File I/O, events, notifications, etc.

- Can create custom WinRT-style components for use by all 3 languages
  - Such components can access native level functionality for any valid WinRT API
  - Except access to the screen...
App Execution Environment

Your App

Game.exe

App Container: Signed & Validated code

WinRT APIs

Core OS

Direct API calls

Brokered API calls

Broker

AppXManifest
<table>
<thead>
<tr>
<th>Win32</th>
<th>WRL</th>
<th>WinRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoCreateInstance</td>
<td>ComPtr&lt;IObject&gt;</td>
<td>Foo^ foo = ref new Foo()</td>
</tr>
<tr>
<td>QueryInterface</td>
<td>foo.As(&amp;bar)</td>
<td>N/A</td>
</tr>
<tr>
<td>AddRef / Release</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>N/A</td>
<td>std::vector</td>
<td>Platform::Array&lt;&gt;^</td>
</tr>
</tbody>
</table>
Classic COM

```c
{
    IPointerPoint *spPointerPoint;
    hr = args->get_CurrentPoint(&spPointerPoint);

    IPointerDevice *spPointerDevice;
    spPointerPoint->get_PointerDevice(&spPointerDevice);

    PointerDeviceType deviceType;
    spPointerDevice->get_PointerDeviceType(&deviceType);

    ...

    spPointerPoint->Release();
    spPointerDevice->Release();
}
```
C++/Cx

{
    auto pointerPoint = args->CurrentPoint;
    auto pointerDevice = pointerPoint->PointerDevice;
    auto deviceType = pointerDevice->PointerType;

    ...
}

Visual Studio

- Updated C++ language support
- File->New Project templates for native DirectX C++ apps
- DirectX HLSL shader compilation and syntax highlighting
- Packaging compiled HLSL shaders into the .appx package
- Support for other asset types in MSBuild and previewer
- Visualization, processing and packaging of
  - Textures, meshes, shaders, and audio
- Debugging DirectX API calls
- Separate talk: Sponsored Session
```
#include <ppl.h>
#include <ppltasks.h>

using namespace Concurrency;

float f = 1.0f;
task<int>([=]() { return foo(f); }).then([&](int x) { bar(x); }).then(baz);
```
App Model

- App Activation
  - The initiation/launch sequence
- Event Handlers
  - Replace message pump switch statement
- Process Lifetime Management
  - Power efficiency
- App Container
  - Enables app sandboxing
- App Package
  - Setup for OS installer, manifest
App Activation

- User initiates app launch
- OS opens **splash screen** from package and displays it
- OS loads app code, and calls methods (IFrameWorkView)

```cpp
DirectxApp::Initialize
DirectxApp::SetWindow
DirectxApp::Load
DirectxApp::Run
DirectxApp::Uninitialize
```
ref class MyApp : public IFrameworkView
{
public:

    MyApp();

    // IFrameworkView Methods
    virtual void Initialize(CoreApplicationView^ applicationView);
    virtual void SetWindow(CoreWindow^ window);
    virtual void Load(String^ entryPoint);
    virtual void Run();
    virtual void Uninitialize();
}
App Model – Activation

- **Initialize()**
  - Initialize app member variables
  - Register for application-wide events

- **SetWindow()**
  - Save app window for later use
  - Register for window events

- **Load()**
  - Determine app behavior based on EntryPoint string
  - Begin asynchronous loading of app resources
    - Prioritize loading splash screen resources first
void MyApp::SetWindow(CoreWindow^ window)
{
    window->SizeChanged +=
        ref new TypedEventHandler<CoreWindow^, WindowSizeChangedEventArgs^>(
            this, &MyApp::OnWindowSizeChanged);

    m_renderer->Initialize(window);
}
App Model – Activation

- Run()
  - Display loading screen
  - Run game loop
  - Process window messages

- Uninitialize()
  - Release resources cleanly
    - No action required for ComPtr or WinRT types
void MyApp::Run()
{
    auto dispatcher = CoreWindow::GetForCurrentThread()->Dispatcher;

    while (!m_windowClosed)
    {
        dispatcher->ProcessEvents(CoreProcessEventsOption::ProcessAllIfPresent);
        m_renderer->Update();
        m_renderer->Render();
        m_renderer->Present();
    }
}
App Activation

Initialize()
SetWindow()
Load()
Run()
Uninitialize()
App Activation Timeline

- **Splash Screen**: OS Loads your App
- **Loading Screen**: App Loads Loading Screen
- **Menu Screen**: App is running
- **Gameplay**: Load()Run()
Packaging and Manifest

- APPX file is a ZIP file
- Installed by Windows using WU technology
- Manifest describes capabilities of app to user
- User can disable capabilities with OS settings pane
- Installation is an OS process, no arbitrary code run at install time
Process Lifetime Management

- Suspend/Resume

- User Launches App
- Running App: Suspended App
- App gets 5 seconds to work after suspend message
- Suspending
- Resuming
- Apps are notified when they have been resumed
- Suspended App: Low Memory
- Apps are not notified before they are terminated
- Terminated App
Process Lifetime Management

```cpp
void MyApp::Initialize(CoreApplicationView^ coreAppView)
{
    CoreApplication::Suspending += ref new
        EventHandler<SuspendingEventArgs^>(this, &MyApp::OnSuspending);
}

void MyApp::OnSuspending(Object^ sender, SuspendingEventArgs^ args)
{
    auto deferral = args->SuspendingOperation->GetDeferral();
    SaveMyStateAsync().then([=]() {
        deferral->Complete();
    });
}
```
Presentation Model
Presentation Model

- Window Size/Resize
- Layout
  - Full screen, filled, snapped
- Rotation
  - Specify which orientations are supported
- Window Activation
  - Focus
- Entered/Exited
  - Useful for mouse
View State: Fullscreen, Filled, and Snapped

- Full screen
  16 x 9

- Filled
  4 x 3

- Snapped
  Phone-like

- By default, app will receive resize events
- Running games can pause if snapped
- Users can have a game at 4 x 3 and their social app snapped
void MyApp::SetWindow(CoreWindow^ window)
{
    window->SizeChanged +=
        ref new TypedEventHandler<CoreWindow^,WindowSizeChangedEventArgs^>(
            this, &MyApp::OnWindowSizeChanged);

    ApplicationView::GetForCurrentView()->ViewStateChanged +=
        ref new TypedEventHandler
            <ApplicationView^, ApplicationViewStateChangedEventArgs^>(
            this, &MyApp::OnViewStateChanged);
}
Display Orientation

- By default: app will receive resize events
- App can lock rotation at a specific orientation
  - Property can be set in manifest, and at runtime
Decision: Rotation

Let OS Rotate:
- Recommended for most apps
- May be disruptive of some types of gameplay

  - Recommended for
    - Event-based games
      - *Solitaire, Mahjong*
    - Any game that is paused or at a menu
    - Non-game apps

Block Rotation by OS:
- No potential for glitching gameplay
- If user rotates
  - OS UI is invoked from different edge
  - Notifications may be sideways or inverted

  - Recommended for
    - Full-real time games
      - For example, Arcade games
    - Accelerometer/gyro games
      - For example, *Marble Maze*
    - Augmented Reality Apps
Rotation Preferences

{
    DisplayOrientations::None;          // Enable rotation by OS/Accelerometer
    DisplayOrientations::Landscape;     // Lock rotation by OS/Accelerometer
    DisplayOrientations::LandscapeFlipped;  // and enable this orientation
    DisplayOrientations::Portrait;
    DisplayOrientations::PortraitFlipped;
}

using namespace Windows::Graphics::Display;

DisplayProperties::AutoRotationPreferences = DisplayOrientations::Landscape
                                              | DisplayOrientations::LandscapeFlipped;
void MyApp::OnWindowActivationChanged(
    CoreWindow^ sender,
    WindowActivatedEventArgs^ args)
{
    auto state = args->WindowActivationState;

    if(state == CoreWindowActivationState::Deactivated)
        OutputDebugString("Focus Lost");

    if(state == CoreWindowActivationState::CodeActivated ||
        state == CoreWindowActivationState::PointerActivated)
        OutputDebugString("Focus Regained");
}
void MyApp::OnPointerEntered(
    CoreWindow^ window,
    PointerEventArgs^ args)
{
    // turn mouse cursor off (hidden)
    window->PointerCursor = nullptr;
}
Pixel Density

- Handle DPI for graphics (XAML is fairly resolution independent)
- Be careful of touch controls. Users hands are same size in inches independent of monitor size.

- 1080p panels may be 10” to 30” to 75”
void MyApp::OnSuspending(Object^ sender, SuspendingEventArgs^ args)
{
    SuspendingDeferral^ deferral = args->SuspendingOperation->GetDeferral();

task<void>([=]()
{
    auto localState = ApplicationData::Current->LocalSettings->Values;
    auto roamingState = ApplicationData::Current->RoamingSettings->Values;
    localState->Insert(
        "GameTime", PropertyValue::CreateSingle(m_gameTime));
    roamingState->Insert(
        "MaxLevel", PropertyValue::CreateUInt32(m_maxLevelUnlocked));
}).then([=]()
{
    deferral->Complete();
});
State Persistence – Loading

```cpp
void MyApp::Load(String^ entryPoint)
{
    LoadMyGameStateAsync().then( [=]() {
        auto localState = ApplicationData::Current->LocalSettings->Values;
        auto roamingState = ApplicationData::Current->RoamingSettings->Values;
        m_gameTime = safe_cast<IPropertyValue^>(localState->Lookup("GameTime"))->GetSingle();
        m_maxLevelUnlocked = safe_cast<IPropertyValue^>(roamingState->Lookup("MaxLevel"))->GetUInt32();
    }).then( [=]() {
        m_loadingComplete = true;
    });
}
```
File Input / Output

task<byte*> LoadSkyAsync()
{
    auto folder = Package::Current->InstalledLocation;
    return task<StorageFile^>(folder->GetFileAsync("sky.dds")).then([](StorageFile^ file){
        return FileIO::ReadBufferAsync(file);
    }).then([](IBuffer^ buffer){
        auto fileData = ref new Array<byte>(buffer->Length);
        DataReader::FromBuffer(buffer)->ReadBytes(fileData);
        return fileData->Data;
    });
}
...
LoadSkyAsync().then([=](byte* skyTextureData)
{
    CreateTexture(skyTextureData);
    m_loadingComplete = true;
});
Break

resume at 2:00 pm
Metro Features
**Live Tile**

- Dynamic content on Start screen tile
- Use for sending informational updates
- Typical information
  - What level player is on
  - Number of friends playing
  - Time until raid starts
- Defined in XML
  - Locally defined
  - Pushed from Microsoft server (WNS)
- Variety of templates to use
  - Custom image for full flexibility

Johnny wants to Surf!
Connect tubes to get him to the ocean!
Live Tile – Example XML

<tile>
  <visual version="1" lang="en-US">
    <binding template="TileWideText03">
      <text id="1">Hello World!</text>
    </binding>
  </visual>
</tile>
Notifications

- Immediate notifications, even from background
- Use for sending urgent updates
- Typical information
  - Player is under attack
  - Friend has invited you to play
  - Raid is starting
- Defined in XML
  - Locally defined
  - Pushed from Microsoft server (WNS)
- Variety of templates to use
Notifications – Example XML

<toast>
  <visual lang="en-US">
    <binding template="ToastText04">
      <text id="1">Big Text</text>
      <text id="2">Small Text 1</text>
      <text id="3">Small Text 2</text>
    </binding>
  </visual>
</toast>
App Bar

- Gesture-triggered menu
- Use this as a pause mechanism
- Simple event to hook
  - No templates
void MyApp::SetWindow(CoreWindow^ window)
{
    EdgeGesture::GetForCurrentView()->Starting += ref new TypedEventHandler<EdgeGesture^, EdgeGestureEventArgs^>(
        [=](EdgeGesture^ gesture, EdgeGestureEventArgs^ args)
        {
            if(args->Kind == EdgeGestureKind::Touch)
                OutputDebugString("App Bar Opening\n");
        });

    EdgeGesture::GetForCurrentView()->Completed += ref new TypedEventHandler<EdgeGesture^, EdgeGestureEventArgs^>(
        [=](EdgeGesture^ gesture, EdgeGestureEventArgs^ args)
        {
            if(args->Kind == EdgeGestureKind::Touch)
                OutputDebugString("App Bar Opened\n");
        });
}
Settings Pane

- Universal OS settings menu
- OS-provided settings
  - Disable internet connection
- App-provided settings
  - Provide options based on state
Settings Pane – Example Code

```c++
SettingsPane::GetForCurrentView()->CommandsRequested +=
    ref new TypedEventHandler<SettingsPane^, SettingsPaneCommandsRequestedEventArgs^>(
        this, &MyApp::OnCommandsRequested);

void MyApp::OnCommandsRequested(SettingsPane^ sender,
    SettingsPaneCommandsRequestedEventArgs^ args)
{
    args->Request->ApplicationCommands->Append(
        ref new SettingsCommand(
            "difficulty", // setting token string
            "Set Difficulty", // user-visible string
            ref new UICommandInvokedHandler(this, &MyApp::OnTestSettingActivated)
        ));
}

void MyApp::OnTestSettingActivated(IUICommand^ command)
{
    ShowDifficultySelectionMenu();
}
```
Share Contract
DataTransferManager::GetForCurrentView()->DataRequested +=
ref new TypedEventHandler<DataTransferManager^, DataRequestedEventArgs^>(
    this, &MyApp::OnDataRequested);

void MyApp::OnDataRequested(
    DataTransferManager^ sender,
    DataRequestedEventArgs^ args)
{
    auto requestData = args->Request->Data;
    requestData->Properties->Title = "High Score";
    requestData->Properties->Description = "DaggerFool47’s High Score";
    requestData->SetText("I got a new high score in MarbleMaze - 26.03 seconds!");
}
Input

- Windows 8 is “Touch First” so please start with touch

- Then enable as many forms of input as make sense for your app
  - It’s okay to have multiple ways to do the same thing

- Make the input forms work well together
  - For example, our samples add up control inputs from different sources so they can be used in any combination
Virtual Touch Thumbsticks
m_gestures = ref new GestureRecognizer();

m_gestures->ManipulationUpdated +=
    ref new TypedEventHandler<GestureRecognizer^, ManipulationUpdatedEventArgs^>
    (this, &MyApp::OnManipulationUpdated);

m_gestures->ProcessInertia();  // Update input

void MyApp::OnPointerPressed(CoreWindow^ sender, PointerEventArgs^ args)
{
    m_gestures->ProcessDownEvent(args->CurrentPoint);
}
Sensor Input

- Gyrometer
- Sensor fusion
Reading orientation sensor data

using Windows::Devices::Sensors;

// Get current reading from sensor
OrientationSensorReading^ orientationReading = m_orientationsensor->GetCurrentReading();
SensorQuaternion^ quat = orientationReading->Quaternion;

// Transform quaternion from device orientation space to world space
// Orientation space is Z-up, right-handed coordinate system
// World space is Y-up, left-handed coordinate system
XMVECTOR orientationQuat = XMVectorSet(-quat->X, quat->Z, quat->Y, -quat->W);

// Create a rotation matrix from the quaternion
// This matrix can be used to rotate an object inside the scene to match
// the rotation of the device
XMMATRIX rotXMMatrix = XMMatrixRotationQuaternion(orientationQuat);
Mouse and Keyboard Input

- Touch First!
- By default, mouse inputs are handled same as touch
- App can detect actual device (mouse, stylus, touch)
- Can register handlers for keyboard events
- Or poll by using GetAsyncKeyState()
Polling for Keyboard Input

// Arrow keys or WASD example
auto upKeyState = window->GetKeyAsyncState(VirtualKey::Up);
auto wKeyState = window->GetAsyncKeyState(VirtualKey::W);

if (upKeyState & CoreVirtualKeyStates::Down ||
    wKeyState & CoreVirtualKeyStates::Down)
{
    m_playerPosition.y += 1.0f;
}
Game Controller Input

- Windows 8 supports Xbox360 –compatible controllers

- Check out new ControllerSketch sample
  - Demonstrates game controller usage from JavaScript app
if ( m_xinputState.Gamepad.wButtons & XINPUT_GAMEPAD_A )
{
    m_aButtonWasPressed = true;
}
else if ( m_aButtonWasPressed )
{
    m_aButtonWasPressed = false; // Trigger once, only on button release
    TriggerSoundEffect();
}
Game Controller D-pad Input

```c
SHORT thumbLeftX = inputState.Gamepad.sThumbLX;
if (abs(thumbLeftX) < XINPUT_GAMEPAD_LEFT_THUMB_DEADZONE)
    thumbLeftX = 0;

SHORT thumbLeftY = inputState.Gamepad.sThumbLY;
if (abs(thumbLeftY) < XINPUT_GAMEPAD_LEFT_THUMB_DEADZONE)
    thumbLeftY = 0;

combinedTiltX += (float)thumbLeftX / 32768.0f;
combinedTiltY += (float)thumbLeftY / 32768.0f;
```
Sound Effects

// Create the XAudio2 engine and mastering voice on the default audio device
XAudio2Create(&m_audioEngine);
m_audioEngine->CreateMasteringVoice(&m_masteringVoice);

// Load all audio data for the sound effect into a single in-memory buffer
MediaLoader soundFile(ref new Platform::String(SOUND_FILE));
m_soundEffectBuffer = soundFile.ReadAll();

// Create a single source voice for a sound effect
m_audioEngine->CreateSourceVoice(&m_sourceVoice,
        &(soundFile.GetOutputWaveFormatEx()));

// Trigger sound effect: queue in-memory buffer for playback and start the voice
XAUDIO2_BUFFER buf = {0};
buf.AudioBytes = m_soundEffectBuffer.size();
buf.pAudioData = &m_soundEffectBuffer[0];
buf.Flags = XAUDIO2_END_OF_STREAM;
m_sourceVoice->SubmitSourceBuffer(&buf);
m_sourceVoice->Start();
Streaming Audio and Video

- XAudio2 is the API for game sound effects
  - Included in the OS and the Windows 8 Metro SDK, no separate SDK downloads required
Windows Runtime Media Platform

Windows Runtime (WinRT)

Playback/Preview | Capture | Transcode | Streaming | Extensibility | Protection | MediaControl

Audio/Video Source | Video Decoder | Video Effect 1 | Video Effect 2 | Video Encoder | Audio/Video Sink

Audio Decoder | Audio Effect 1 | Audio Effect 2 | Audio Encoder

Media Foundation

Direct3D | Windows Audio Session API (WASAPI)
Windows Runtime Media Platform
Multiplayer Metro style

- Server Options
  - WNS, ASP, WCF, Azure
  - WebSockets – now a web standard
- Client Options
  - IPv4 TCP/UDP
  - IPv6 – much better NAT traversal, future proofing
- Scenarios
  - Cloud-based server, peer to peer, etc.
- Best Practices
Cloud Server (MMO)

- Variety of server hosting options available
  - Azure, etc.
- Bandwidth is not that expensive
- Cloud APIs are easy to use
  - WNS, ASP, WCF
  - WebSockets
    - Proximity API provided in all languages, including for browser
    - Becoming a standard (RFC complete)
Peer to Peer (Arcade, FPS)

- Reduces your bandwidth bills
- Enables lower-latency for intensely interactive titles
  - Arcade-like level of responsiveness
  - Can validate via multipath for security
- Discovery via Proximity API
  - Detects via tap/contact, Bluetooth, Wi-Fi Direct, NFC, etc.
- Run IPv4 or IPv6 sockets once in-game
- “Local Server” model works well
  - Migrate between clients onSuspending
In-Package Server

- If implemented as a Metro style app, it can be suspended after inactivity
- If this is a classic app, it must be on a machine no Metro client is on
  - Due to loopback blocking of that client
In-Package Server

- If implemented as a Metro style app, it can be suspended after inactivity
- If this is a classic app, it must be on a machine no Metro client is on
  - Due to loopback blocking of that client
In-Package Server

- If implemented as a Metro style app, it can be suspended after inactivity
- If this is a classic app, it must be on a machine no Metro client is on
  - Due to loopback blocking of that client
Best Practices

- If starting fresh, or using JScript
  - Use WebSockets to get up and running

- If you have an existing TCP/UDP codebase
  - IPv4 code will still need STUN/TURN for NAT traversal
  - IPv6 is easier
  - Note: WinRT threading model is different
    - PPL Async pattern works well

- Watch out for loopback blocking
  - Loopback allowed on dev machines (e.g. by VS) only
Networking Docs Links

- Developing Connected Applications

- Websockets

- Sockets

- Proximity & Discovery
3D Graphics via Direct3D 11

- 3D Graphics uses the same API as on Desktop/Classic
- WinRT API set includes only the latest DirectX 11 syntax
- DirectX 11 API supports multiple hardware generations via *FeatureLevels*
  - Feature_Level_9, Feature_Level_10, Feature_Level_11
  - See next presentation for details
- Direct3D 11 updated for Windows 8
  - See yesterday’s Advanced Direct3D talk 😊
Direct3D 11.1

- Now supports 3D stereo
- Logic ops in output merger
- Dynamic updates to constant Bbuffers
- Low-precision math: min16_float, min10_float, min16_int
  - Hardware can promote silently to single precision float
- UAVs at every stage, not just pixel and compute shaders
- Double precision division instructions
- Video support
  - Graphics + Video integrated driver
  - MSAD4 instruction
- Headless/Session 0
How to Target Frame Rates <> 60Hz

- Metro style apps always have vsync enabled
- FPS will always be one of:
  - 1  60Hz  16.66666ms
  - 2  30Hz  33.33333ms
  - 3  20Hz  50ms
  - 4  15Hz  66.66666ms
  - ...

Present1( /* SyncInterval = */ 2, ... );  // set to 30Hz
XAML UI Toolkit

- Retained-Mode UI Rendering
- Automatic layout
- Interoperable with DirectX
  - `<SwapChainBackgroundPanel>`
  - `<ImageBrush ImageSource=[DXGI Surface]>`
Distribution – Windows Store

- **Download Service:**
  - Download/install your app - updates via patches with 64 KB deltas
  - Max package size is 2 GB
  - Trials, free apps

- **Billing Service:**
  - App prices from $1.49 to $999.99, and in-app purchases
  - Revenue percentage: 70/30 - 80/20 after $25,000
  - Option for 3rd party billing - with no fee from Microsoft

- Developer membership: $99 per 2 years
- Package certification tool provided in the Windows 8 Metro SDK
  - Store emulator likewise
Pro Dev Hints

- Set up a multi-monitor PC with a touch monitor as the primary
- Set a breakpoint in the DirectX throwException macro in DirectSample.h
- Hook WindowClosed event
- Use Alt + F4 to test termination code path
  - After 5 seconds of being occluded, app gets a Suspend event
  - Then may get terminated
  - App can also be explicitly terminated via touch
    - Drag from top to bottom
Three Steps to Success

1. Download the Windows 8 Metro SDK and Visual Studio 11

2. Start writing your App

3. Leverage the materials
   From GDC 2012 and BUILD on MSDN

4. Ship it
Related Sessions and Materials

- More Developer Day Talks
  - Developing Metro-Style Games on a Full Range of Windows 8 PCs
  - Dan Mclachlan - next timeslot

- GDC Sessions
  - Monetization Strategies for Windows 8 - Shai Hinitz
    - Wednesday 11:00-12:00 Room 2024
  - Visual Studio for Game Developers - Boris Jabes
    - Wednesday 2:00-3:00 Room 2024
  - Developing a Great Metro Style Game for Windows 8
    - Thursday 10:00-11:00 Room 2024
Summary of Resources

- Introductory BUILD Talks
  - Describe the platform and philosophy
  - Samples for Metro style app features
    - Samples are mostly in C#, so some porting is needed
  - Dedicated talks on graphics
  - Game Curriculum:
BUILD Game Curriculum

- **PLAT-750T Build your first Metro style game**
  - APIs for writing games in the Metro style. How to use Direct3D in CoreWindow

- **PLAT-751T 3D Graphics for Metro style Apps and Games**
  - Introduction to Direct3D for new developers. How to use Direct3D in the Metro style.

- **PLAT-752T Tuning GPU Usage for any Form Factor**
  - How to scale Direct3D graphics content across the GPU performance range.

- **TOOL-761T A Lap Around DirectX Game Development**
  - Demonstration of the new tools for viewing/debugging DirectX assets in Visual Studio 11.

- **PLAT-754T From Touch to Gamepads: Master Player Input in your Metro style Game**
  - The various input methods you can enable in your Metro style game.

- **PLAT-755T Compelling Audio and Video for Metro style Games**
  - The APIs provided by the Windows Runtime for audio and video integration.

- **TOOL-690C Under the Covers with C++ for Metro style Apps**
  - Creating your own WinRT projections in C++.
<table>
<thead>
<tr>
<th>Topic</th>
<th>API</th>
<th>Sample App</th>
<th>BUILD PPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initializing Direct3D</td>
<td>D3D</td>
<td>D3DTutorial</td>
<td>751</td>
</tr>
<tr>
<td>Handling system events</td>
<td>CoreWindow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persisting state</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metro app features (tiles, notifications, app bar)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rescaling screen size to optimize performance</td>
<td>D3D</td>
<td>D3DPostProcessing</td>
<td></td>
</tr>
<tr>
<td>Touch Thumbsticks</td>
<td>CoreWindow</td>
<td>Simple3DTouch</td>
<td></td>
</tr>
<tr>
<td>Relative Mouse Movement</td>
<td>CoreWindow</td>
<td>Simple3DTouch</td>
<td></td>
</tr>
<tr>
<td>Joystick/Gamepad input</td>
<td>XInput</td>
<td>Simple XInput Controller</td>
<td>754</td>
</tr>
<tr>
<td>Game Audio</td>
<td>XAudio2</td>
<td></td>
<td>755</td>
</tr>
<tr>
<td>Networking</td>
<td>Sockets</td>
<td></td>
<td>580</td>
</tr>
<tr>
<td>Scaling across hardware performance range</td>
<td>D3D</td>
<td>Simple3DSprites</td>
<td></td>
</tr>
<tr>
<td>Stereo 3D</td>
<td>D3D</td>
<td>Simple3DGameDX</td>
<td></td>
</tr>
<tr>
<td>XAML / Direct3D Interop</td>
<td>XAML</td>
<td>Simple3DGameXAML</td>
<td></td>
</tr>
</tbody>
</table>