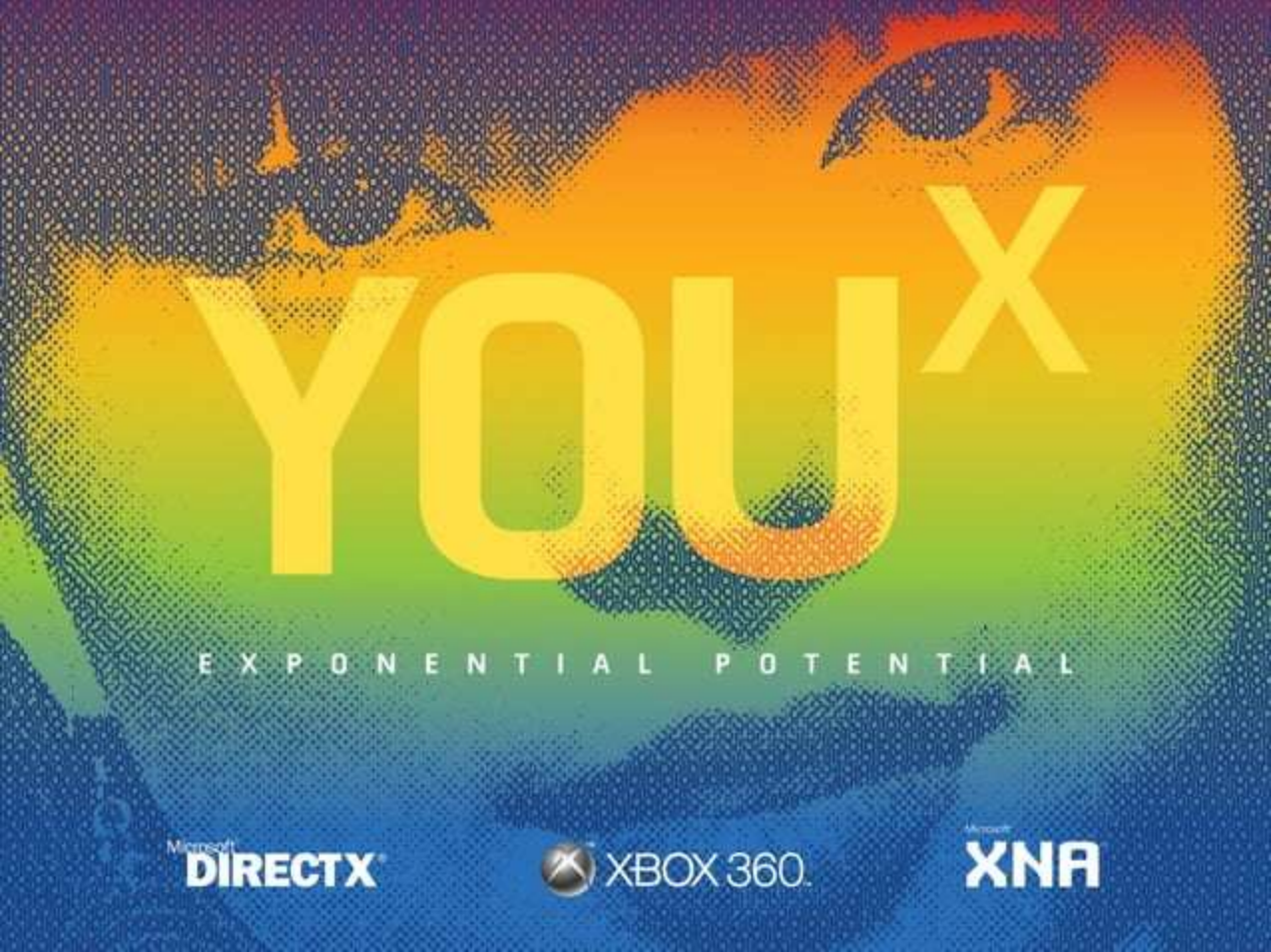

YOU X
EXPONENTIAL POTENTIAL
Microsoft DIRECTX
XBOX 360
Microsoft XNA

# Next-Generation Graphics Programming on Xbox 360

# Prerequisites

- Basic system architecture
  - Caches, Bandwidth, etc.
- Basic Direct3D
  - Drawing, render targets, render state
  - Vertex and pixel shaders

# Overview

- Xbox 360 System Architecture
- Xbox 360 Graphics Architecture Details
- Direct3D on Xbox 360
- Xbox 360 graphics APIs
- Shader development
- Tools for graphics debugging and optimization

# Xbox 360

- 512 MB system memory
- IBM 3-way symmetric core processor
- ATI GPU with embedded EDRAM
- 12x DVD
- Optional Hard disk

# The Xbox 360 GPU

- Custom silicon designed by ATi Technologies Inc.
- 500 MHz, 338 million transistors, 90nm process
- Supports vertex and pixel shader version 3.0+
  - Includes some Xbox 360 extensions

# The Xbox 360 GPU

- 10 MB embedded DRAM (EDRAM) for extremely high-bandwidth render targets
  - Alpha blending, Z testing, multisample antialiasing are all free (even when combined)
- Hierarchical Z logic and dedicated memory for early Z/stencil rejection
- GPU is also the memory hub for the whole system
  - 22.4 GB/sec to/from system memory

# More About the Xbox 360 GPU

- 48 shader ALUs shared between pixel and vertex shading (unified shaders)
  - Each ALU can co-issue one float4 op and one scalar op each cycle
  - Non-traditional architecture
- 16 texture samplers
- Dedicated Branch instruction execution

# More About the Xbox 360 GPU

- 2x and 4x hardware multi-sample anti-aliasing (MSAA)
- Hardware tessellator
  - N-patches, triangular patches, and rectangular patches
- Can render to 4 render targets and a depth/stencil buffer simultaneously

# GPU: Work Flow

- Consumes instructions and data from a command buffer
  - Ring buffer in system memory
  - Managed by Direct3D, user configurable size (default 2 MB)
  - Supports indirection for vertex data, index data, shaders, textures, render state, and command buffers
- Up to 8 simultaneous contexts in-flight at once
  - Changing shaders or render state is inexpensive, since a new context can be started up easily

# GPU: Work Flow

- Threads work on units of 64 vertices or pixels at once
- Dedicated triangle setup, clipping, etc.
- Pixels processed in 2x2 quads
- Back buffers/render targets stored in EDRAM
  - Alpha, Z, stencil test, and MSAA expansion done in EDRAM module
- EDRAM contents copied to system memory by “resolve” hardware

# GPU: Operations Per Clock

- Write 8 pixels or 16 Z-only pixels to EDRAM
  - With MSAA, up to 32 samples or 64 Z-only samples
- Reject up to 64 pixels that fail Hierarchical Z testing
- Vertex fetch sixteen 32-bit words from up to two different vertex streams

# GPU: Operations Per Clock

- 16 bilinear texture fetches
- 48 vector and scalar ALU operations
- Interpolate 16 float4 shader interpolants
- 32 control flow operations
- Process one vertex, one triangle
- Resolve 8 pixels to system memory from EDRAM

# GPU: Hierarchical Z

- Rough, low-resolution representation of Z/stencil buffer contents
- Provides early Z/stencil rejection for pixel quads
- 11 bits of Z and 1 bit of stencil per block

# GPU: Hierarchical Z

- NOT tied to compression
  - EDRAM BW advantage
- Separate memory buffer on GPU
  - Enough memory for 1280x720 2x MSAA
- Provides a big performance boost when drawing complex scenes
  - Draw opaque objects front to back

# GPU: Xbox Procedural Synthesis

- Dedicated hardware channel between CPU L2 cache and GPU
  - 10.8 GB/sec
- Useful for dynamic geometry, instancing, particle systems, etc.
- Callbacks inserted into command buffer spin up XPS threads on CPU during rendering

# GPU: Xbox Procedural Synthesis

XPS threads run your supplied code

- GPU consumes data as CPU generates it
  - Separate, small command buffer in locked L2 cache section

# GPU: Hardware Tessellation

- Dedicated tessellation hardware
- Occurs before vertex shading
- Does not compute interpolated vertex positions
  - Parametric values are sent to a special vertex shader designed for tessellation

# GPU: Hardware Tessellation

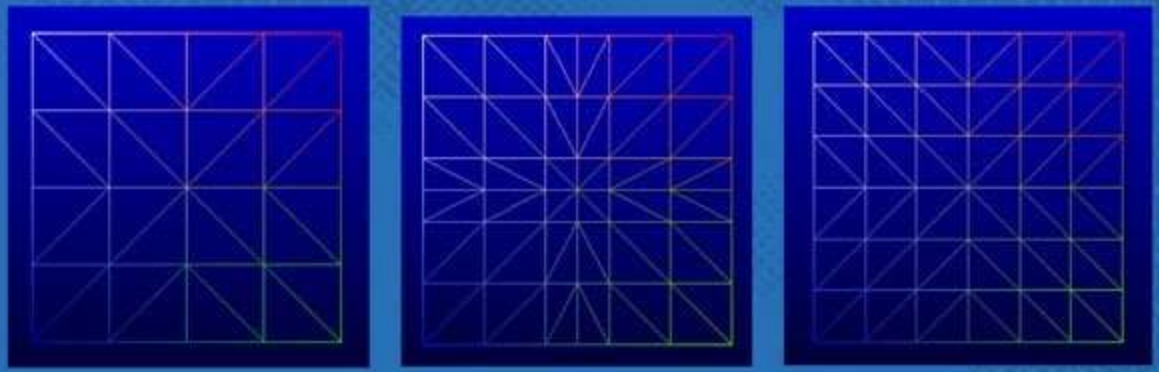
- 3 modes of tessellation
  - Discrete: integer parameter, triangles only
  - Continuous: float parameter, new vertices are shifted to their final position
  - Per-edge: Special index buffer used to provide per-edge factors, used for patches only, cannot support traditional index data as well

# Discrete Tessellation Sample

- Triangle with tessellation level 4

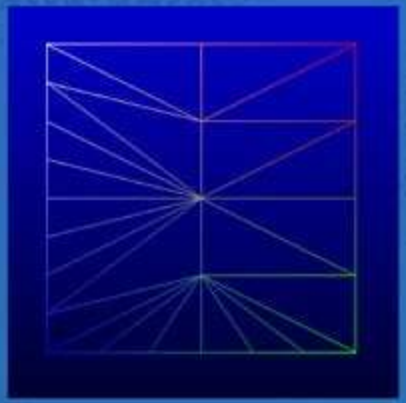
# Continuous Tessellation Sample

- Quad with tessellation levels 3, 4, 5



# Per-Edge Tessellation Sample

- Quad-patch with per-edge tessellation factors of 1, 3, 5, and 7



# GPU: Textures

- 16 bilinear texture samples per clock
  - 64bpp runs at half rate, 128bpp at quarter rate
  - Trilinear at half rate
- Unlimited dependent texture fetching
- DXT decompression has 32 bit precision
  - Better than Xbox (16-bit precision)

# GPU: Textures

- Texture arrays – generalized version of cube maps
  - Up to 64 surfaces within a texture, optional MIPmaps for each surface
  - Surface is indexed with a [0..1] z coordinate in a 3D texture fetch
- Mip tail packing
  - All MIPs smaller than 32 texels on a side are packed into one memory page

# GPU: Resolve

- Copies surface data from EDRAM to a texture in system memory
- Required for render-to-texture and presentation to the screen
- Can perform MSAA sample averaging or resolve individual samples
- Can perform format conversions and biasing

# Direct3D 9+ on Xbox 360

- Similar API to PC Direct3D 9.0
- Optimized for Xbox 360 hardware
  - No abstraction layers or drivers—it's direct to the metal
  - Exposes all Xbox 360 custom hardware features
    - New state enums
    - New APIs for finer-grained control and completely new features

# Direct3D 9+ on Xbox 360

- Communicates with GPU via a *command buffer*
  - Ring buffer in system memory
- Direct Command Buffer Playback support

# Direct3D: Drawing

- Points, lines, triangles, quads, rects, polygons
  - Stripped lines, triangles, quads, polygons
  - Configurable reset index to reset strips within an index buffer
- DrawPrimitive, DrawIndexedPrimitive
- DrawPrimitiveUP, DrawIndexedPrimitiveUP
  - No vertex buffers or index buffers required

# Direct3D: Drawing

- BeginVertices / EndVertices
  - Direct allocation from the command buffer
  - Better than DrawPrimitiveUP
- SetStreamSourceFrequency
  - Not directly supported on Xbox 360
  - Can be done with more flexibility in vertex shader
- RunCommandBuffer
  - Direct support for modifying and playing command buffers

# Direct3D: State

- Render state and sampler state
  - New states added for Xbox 360
    - Half-pixel offset
    - High-precision blending
    - Primitive reset index
  - Unsupported states ignored by Direct3D

# Direct3D: State

- Lazy state
  - Direct3D batches up state in blocks that are optimized for submission to the GPU
  - Uses special CPU instructions for very low CPU overhead
- State API extensions for Xbox 360
  - SetBlendState
  - Individual blend control for each render target

# Direct3D: Render Targets

- New format for high dynamic range
  - 2:10:10:10 float
  - 7 bits mantissa, 3 bits exponent for each of R, G, B
  - No matching texture format
  - Special mode which allows for 10 bits of alpha precision on blends
- Optional sRGB gamma correction (2.2) on 8:8:8:8 fixed point

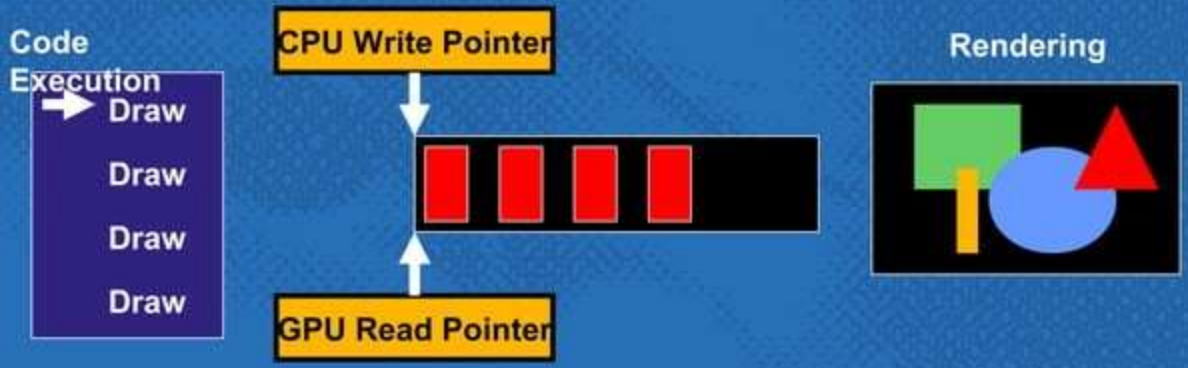
# Direct3D: New Texture Formats

- DXN
  - 2-component format with 8 bits of precision per component
  - Great for normal maps
- CTX1
  - Also good for normal maps
  - Less bits per pixel than DXN

# Direct3D: New Texture Formats

- DXT3A, DXT5A
  - Single component textures made from a DXT3/DXT5 alpha block
  - 4 bits of precision

# Direct3D: Command Buffer




- Ring buffer that allows the CPU to safely send commands to the GPU
- Buffer is filled by CPU, and the GPU consumes the data

# Direct3D: Predicated Tiling

- EDRAM is not large enough for 720p rendering with MSAA or multiple render targets
- Solution: Predicated tiling
  - Divide the surface into a handful of “tiles”
  - Direct3D replays an accumulated command buffer to each tile
  - Direct3D stitches together a final image by resolving each tile pass to a rect in a texture in system memory

# Direct3D: Predicated Tiling

- Optimization
  - Uses GPU hardware predication and screen-space extents capture
  - Only renders primitives on the tiles where they appear
  - Cuts down significantly on duplicate vertex processing

# Direct3D: Predicated Tiling

- What does it cost?
  - Increased vertex processing cost, for primitives that span more than one tile
  - No pixel processing increase
- Required for 720p and MSAA

# XGraphics

- Graphics API for direct access to Direct3D resources and low-level texture data manipulation
- Create/update resources without the D3D device
  - XGSetTextureHeader, XGSetVertexBufferHeader, XGSetIndexBufferHeader, etc
  - Useful for threaded loading code

# XGraphics

- Direct management of resource memory
  - Use your own allocators, etc.
- Texture tiling functions
- Endian swapping functions
- Win32 library is useful for tool pipelines and offline resource packing

# Shaders

- Two options for writing shaders
  - HLSL (with Xbox 360 extensions)
  - GPU microcode (specific to the Xbox 360 GPU, similar to assembly but direct to hardware)
- Recommendation: Use HLSL
  - Easy to write and maintain
  - Replace individual shaders with microcode if performance analysis warrants it

# High-Level Shading Language (HLSL)

- Xbox 360 extensions
  - Texture sampler modes (offsets, filtering override, etc.)
  - Memory export
  - Predicated tiling screen space offset
  - Vertex fetching/index manipulation
  - Tesselation

# Shaders: GPU Microcode

- Shader language that maps 1:1 with the hardware
- Similar to assembly shaders in appearance
- Somewhat esoteric, but extremely flexible
- Plan for HLSL, write microcode for performance gains where needed

# Shaders: Branching

- Static branching
  - Branching based on bool or int constants
  - Example: Perform texture sample if constant b15 is TRUE
- Dynamic branching
  - Branching based on a computed or interpolated value
  - Example: Compute dot product, light pixel if result > 0

# Shaders: Branching

- Dynamic branching can be as fast as static branching if all vertices/pixels in a 64-unit vector take the same branch

# Shaders: Vertex Declarations and Fetches

- Xbox 360 GPU does not natively support vertex declarations or streams
- Shaders are patched at run time to match the current vertex declaration
  - Vertex fetch instructions used to load vertex data into the GPU

# Shaders: Vertex Declarations and Fetches

- Shader patching code caches the patched code using a shader-decl pointer pair
  - Try not to create many identical vertex declarations
- More GPU Friendly shaders are added as an extension but are not as flexible

# PIX

- The ultimate performance and graphics debugging tool
- Incorporates several modules
  - Performance monitor
  - CPU performance analysis
  - Command buffer capture, playback, and analysis
  - Shader and mesh debuggers

# PIX

- Hardware accelerated
  - GPU has hundreds of performance counters
  - Development kit has special hardware for PIX

# PIX: Performance Monitor

- Scrolling graph display
- Configurable graphs
  - CPU, GPU, memory, storage stats
- Best place to accurately gauge frame rate/frame time
- Runs continuously while PIX is open
- Works with any application running on the development kit

# PIX: CPU Timeline

- Shows the CPU usage of graphics calls
- Also shows the CPU usage of your title's subsystems
- Use PIXBeginNamedEvent / PIXEndNamedEvent APIs to label your subsystems by name and color

# PIX: GPU Timeline

- Shows the execution of each graphics command on the GPU
  - Timing and statistics
  - Full GPU and Direct3D state
  - All input and output resources (textures, surfaces, shaders, etc.)
- Event breakdown
  - Render target views
  - Mesh debugger

# PIX: Shader debuggers

- Vertex shader debugger
  - Click on a vertex in the mesh debugger
  - HLSL And Microcode
- Pixel shader debugger
  - Right-click on a pixel in a render target
  - Select the right draw call that influenced that pixel
  - HLSL and Microcode

# Conclusion

- Direct3D customized and optimized for the hardware
  - New APIs to take advantage of custom features
  - Direct to the metal, no drivers or HAL

# Conclusion

- Extremely powerful graphics hardware
  - Unified shaders
  - Shader model 3.0 plus extensions for Xbox 360
  - Fast EDRAM provides a huge amount of dedicated frame buffer bandwidth
  - Dedicated hardware for tesselation, procedural geometry, and multi-sample anti-aliasing

# Conclusion

- High performance shader development
  - Brand new HLSL compiler for Xbox 360
  - GPU microcode to expose the hardware's abilities
  - FXLite for fast effects

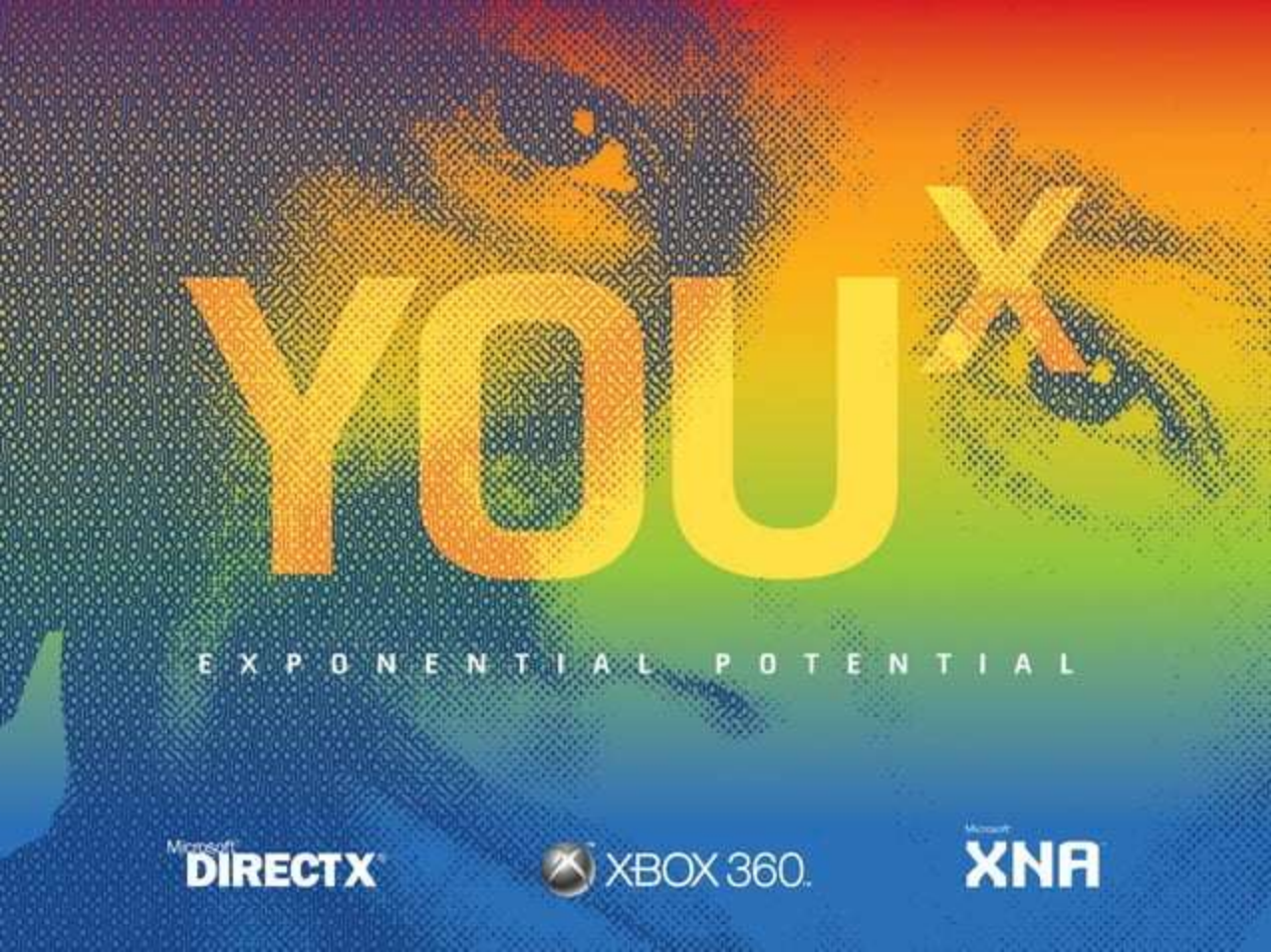
# Conclusion

- Best performance tools in the industry
  - PIX is your best friend and your secret weapon
  - Hardware-accelerated
  - Exposes every bit of performance info

# Resources

- GDC 2006 Presentations
  http://msdn.com/directx/presentations
- DirectX Developer Center
  http://msdn.com/directx
- XNA Developer Center
  http://msdn.com/xna
- XNA, DirectX, XACT Forums
  http://msdn.com/directx/forums
- Email addresses
  directx@microsoft.com (DirectX & Windows development)
  xna@microsoft.com (XNA Feedback)




YOU X
EXPONENTIAL POTENTIAL
Microsoft DIRECTX
XBOX 360
Microsoft XNA