Introduction to XNA; Game Loops

Prof. Aaron Lanterman
School of Electrical and Computer Engineering
Georgia Institute of Technology
Part 1: Introduction to XNA
Dungeon Quest

- Developed in 4 days at the 2007 GDC at the XNA contest
- By Benjamin Nitschke and Christoph Rienaechker

Screenshot from exdream.no-ip.info/blog/2007/07/31/DungeonQuestUpdatedWithSourceCodeNow.aspx
Torpex’s “Schizoid” (on Xbox Live Arcade)

Screenshot from http://screenshots.teamxbox.com/screen/68599/Schizoid/

http://www.gametrailers.com/player/28542.html
XNA GS Framework

• Built on Microsoft’s .NET
  – Makes MS comfortable with letting “ordinary folks” program on the Xbox 360

• C# is standard language for XNA development
  – But in theory could use Managed C++, VB.NET, etc. on the PC
Is managed code too slow for games?

- Vertigo Software ported Quake II to Managed C++, got 85% performance of the original C code
  - Should expect to do better if you have the .NET Common Language Runtime in mind from the beginning

- Xbox 360
  - GPU: 337 million transistors
  - CPU: 165 million transistors
Xbox 360 uses .NET Compact Framework

• Some stuff available in .NET on the PC is missing

• Garbage collector on 360 isn’t as smart as on the PC

• Caused the Schizoid team some trouble, as well as one semester of CS4455
XNA 4.0 requirements

• Windows XP/Vista/7
  – I will be running Windows 7
  – Windows Phone development only works under Windows 7 (not relevant for this class)

• Graphics card supporting at least DirectX 9.0c and Shader Model 2.0
  – Docs say Shader Model 1.1, but that’s iffy
  – HiDef Profiles & Windows Phone development “need” a card supporting at least DirectX 10

From msdn.microsoft.com/en-us/library/bb203925.aspx
XNA 4.0 graphics profiles (1)

- Profiles specify a common set of graphics capabilities
- Reach Profile:
  - PC, Xbox 360, Phone
  - DirectX 9 and Shader Model 2.0
- HiDef Profile:
  - PC, Xbox 360
  - DirectX 10 and Shader Model 3.0

• Some advanced DX 9 cards may luck out
XNA 4.0 graphics profiles (2)

• Reach is a strict subset of HiDef

• Careful: different profiles use different content pipelines

• Can query to see what profiles the user’s hardware supports
  – Only useful on Windows; you know Xbox 360 can handle HiDef and phones can only handle Reach
XNA GS graphics

- XNA is built on top of DirectX 9
  - Not built on MDX or Managed DirectX
  - Specification of DX10 hardware ensures rich feature set, but DX10 API isn’t used!

- DirectX 9 has a fixed function pipeline, but XNA doesn’t!
  - Everything done with shaders
  - XNA has a BasicEffect to get you started
Why no fixed-function pipeline? (1)

In Microsoft’s own words (paraphrased):

• Programmable pipeline is the future
  – Neither Direct3D 10/11 or Xbox 360 have fixed-function pipeline
• Early adopters and customers said cross-platform goal more important than fixed-function pipeline

Why no fixed-function pipeline? (2)

In Microsoft’s own words (paraphrased):

• Fear is someone would start and finish their game using the fixed-function APIs, and then get dozens of errors when they tried to compile it on the Xbox 360

• Better to know your code works on both right from the beginning

Some convenient things about XNA

• Don’t need to mess with Win32-ish boilerplate (opening a window, etc.)
• Easy interfacing with the Xbox 360 controller (for both Windows and Xbox 360)
• Storage (“saved games”) unified between Windows and Xbox 360
  – On Xbox 360, have to associate data with a user profile, put on hard drive or memory card, etc.
  – XNA “emulates” this on Windows

public class SampleGame : Game {
    private GraphicsComponent graphics;

    public SampleGame() {
        this.graphics = new GraphicsComponent();
        this.GameComponents.Add(graphics);
    }

    protected override void Update() {
    }

    protected override void Draw() {
        this.graphics.GraphicsDevice.Clear(Color.Blue);
        this.graphics.GraphicsDevice.Present();
    }

    static void Main(string[] args) {
        using (SampleGame game = new SampleGame()) {
            game.Run();
        }
    }
}
Careful if you’re on Windows x64

• XNA normally targets “AnyCPU”

• Will break when you try to run on x64 machines, since x64 versions XNA framework dlls don’t exist (and probably never will)

• Workaround: Change target to x86
Caveats about Xbox 360 development

• Many TVs cutoff 5-10% of the pixels around the edge
  – Keep text & important info away from there

• Xbox 360 handles post processing and render targets a little differently than the PC

Info from Alistair Wallis, “Microsoft XNA: A Primer,” interview with Benjamin Nitschke
www.gamecareerguide.com/features/328/microsofts_xna_a_.php?page=4
Contests

• See http://www.dreambuildplay.com and http://www.imaginecup.com

• 2012’s contests are already over…

• …but keep on the lookout for the 2013 Dream Build Play & Imagine Cup contests!
XNA Indie Games

• See [http://create.msdn.com](http://create.msdn.com)
• Join the XNA App Hub (formerly Creator’s Club)
  – The XNA App Hub memberships students get free from DreamSpark will let you run games on the 360, but may not let you take part in Indie Games
• Upload your game, rate content (violence, etc.)
• Peer review: confirm content ratings, check quality
• Can sell your game to Xbox 360 users!
  – 150 MB limit
  – 80, 240, or 400 Microsoft Points ($1, $3, or $5)
• Can sell XNA PC Windows games on Steam…
  – …if Valve gives it a thumbs up
Example: A Fading Melody
### XNA CG sales (March 31, 2009)

<table>
<thead>
<tr>
<th>Game</th>
<th>Date</th>
<th>Units</th>
<th>Price (%)</th>
<th>Quantity</th>
<th>Price</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find Teddy</td>
<td>1-Feb-09</td>
<td>2,181</td>
<td>3.2%</td>
<td>400</td>
<td>$338.80</td>
<td>$237.16</td>
</tr>
<tr>
<td>Remote Masseuse</td>
<td>11-Feb-09</td>
<td>55,000</td>
<td>6.4%</td>
<td>200</td>
<td>$8,470.00</td>
<td>$5,929.00</td>
</tr>
<tr>
<td>Exhaust</td>
<td>14-Feb-09</td>
<td>27,256</td>
<td>3.6%</td>
<td>400</td>
<td>$4,791.60</td>
<td>$3,354.12</td>
</tr>
<tr>
<td>Trajectory</td>
<td>20-Feb-09</td>
<td>1,928</td>
<td>3.1%</td>
<td>200</td>
<td>$142.78</td>
<td>$99.95</td>
</tr>
<tr>
<td>Tomato Blaster</td>
<td>22-Feb-09</td>
<td>704</td>
<td>3.6%</td>
<td>400</td>
<td>$121.00</td>
<td>$84.70</td>
</tr>
<tr>
<td>ZoomaRoom</td>
<td>25-Feb-09</td>
<td>4,703</td>
<td>8.5%</td>
<td>200</td>
<td>$963.16</td>
<td>$674.21</td>
</tr>
<tr>
<td>Alchemist</td>
<td>27-Feb-09</td>
<td>1,356</td>
<td>7.4%</td>
<td>200</td>
<td>$244.42</td>
<td>$171.09</td>
</tr>
<tr>
<td>ZP2K9</td>
<td>28-Feb-09</td>
<td>19,628</td>
<td>17.3%</td>
<td>200</td>
<td>$8,194.12</td>
<td>$5,735.88</td>
</tr>
<tr>
<td>Snake360 Lite</td>
<td>10-Mar-09</td>
<td>4,798</td>
<td>7.8%</td>
<td>200</td>
<td>$909.92</td>
<td>$636.94</td>
</tr>
<tr>
<td>Solar</td>
<td>20-Mar-09</td>
<td>8,000</td>
<td>18.3%</td>
<td>200</td>
<td>$3,547.72</td>
<td>$2,483.40</td>
</tr>
<tr>
<td>Clock 24-7</td>
<td>21-Mar-09</td>
<td>4,865</td>
<td>5.1%</td>
<td>200</td>
<td>$602.58</td>
<td>$421.81</td>
</tr>
<tr>
<td><strong>Totals</strong></td>
<td></td>
<td>350,433</td>
<td>25,049</td>
<td></td>
<td>$69,550.80</td>
<td>$48,685.56</td>
</tr>
</tbody>
</table>

Part 2: Game Loops
Credit to where it is due

• Koen Witters
  – Thinking about game loops

• Shawn Hargreaves
  – Details about XNA’s game loop

• Side note: next few slides on game loops contain rough *pseudocode*
Simplest game loop (1)

```java
running = true;

while(running) {
    update();
    draw();
}
```

- Draw() has things like `bad_guy.x += 1;`
- What could possibly go wrong?
Simplest game loop (2)

- Game runs faster on faster hardware, slower on slower hardware
- Less of a problem if hardware is well-defined; Apple II+, Commodore 64, game console
- Try an original Mac game on a Mac II: too fast!
- Big problem on PCs/Macs with varying speed
- Can still be a problem if update time varies from iteration to iteration (i.e. varying number of bad guys)
  - See Defender and Robotron: 2084

http://dewitters.koonsolo.com/gameloop.html
running = true;
seconds_per_frame = 1/60;

while(running) {
    update();
    draw();
    if (seconds_per_frame_not_elapsed_yet)
        wait(remaining_time);
    else {
        oooops! We are running behind!
    }
}

• What could possibly go wrong?
FPS dependent on constant GS (2)

• Slow hardware:
  – If fast enough to keep up with FPS no problem
  – If not: game will run slower
  – Worst case: some times runs normally, sometimes slower – can make unplayable

http://dewitters.koonsolo.com/gameloop.html
• Fast hardware:
  – Wasting cycles on desktops - higher FPS gives smoother experience, why not give that to the user?
  – Maybe not so bad philosophy on mobile devices – save battery life!
  – Also may not be bad if user is wants to run other processes
GS dependent on variable FPS (1)

running = true;

while(running) {
    update(time_elapsed);
    draw();
}

• Use time_elapsed in your state update computations:
  bad_guy.x += time_elapsed * bad_guy.velocity_x;

• What could possibly go wrong?

http://dewitters.koonsolo.com/gameloop.html
GS dependent on variable FPS (2)

- Slow hardware:
  - Game sometimes bogs down, i.e. when lots of stuff is on the screen
    - Slows down player and AI reaction time
  - If time step is too big:
    - Physics simulations may become unstable
    - “Tunneling” (need “swept collision detection”)

http://dewitters.koonsolo.com/gameloop.html
GS dependent on variable FPS (3)

• Fast hardware:
  – Shouldn’t be a problem, right?
  – What could possibly go wrong?
GS dependent on variable FPS (4)

- Fast hardware:
  - More calculations per second for some quantity, more round off errors can accumulate
  - Multiplayer game: players with systems with different speeds will have game states drifting apart
  - Good example:
    - www.nuclex.org/articles/xna-game-loop-basics

http://dewitters.koonsolo.com/gameloop.html
Balancing act

• Want fast update rate…
• …but still be able to run on slow hardware
• Many more possibilities

Photo by Aaron Sneddon; under the Creative Commons Attribution 3.0 Unported license

http://dewitters.koonsolo.com/gameloop.html
Tasks with different granularity

• Run often:
  – Physics engine location & orientation updates
  – 3-D character display

• Run less often:
  – Collision detection
  – Player input
  – Head-up display

• Run even less often:
  – “immediate A.I.”, networking

• Careful: A.I. might be unstable with larger time steps – not just physics!
Example: MotoGP

• Main game logic: 60 updates per second
  – “input, sound, user interface logic, camera movement, rider animations, AI, and graphical effects”

• Physics: 120 updates per second

• Networking: 4 to 30 updates per second, depending on number of players – more players results in less often updates to conserve bandwidth

XNA game loop: fixed step

• Game.IsFixedTimeStep = true; (default)
• XNA calls Update() every “TargetElapsedTime” (defaults to 1/60 seconds)
  • Repeat call as many times as needed to catch up with current frame (in XNA >= 2.0)
• XNA hopefully calls Draw(), then waits for next update
• If Update+Draw time < TargetElapsedTime, we get
  – Update
  – Draw
  – Hang out for rest of time (nice on Windows so other processes can run)

game-timing-in-xna-game-studio-2-0.aspx
XNA may get behind (1)

• Why would Update+Draw time > TargetElapsedTime?
  – Computer slightly too slow
  – Computer way too slow
  – Computer mostly fast enough, but may have too much stuff on screen, big texture load, or garbage collection
  – Paused program in debugger

XNA may get behind (2)

- What happens if Update+Draw time > TargetElapsedTime?
  - Set GameTime.IsRunningSlowly = true;
  - Keep calling Update (without Draw) until caught up
    - Makes sure game is in right state with Draw finally happens
  - If too far behind… punt

When XNA gets behind (1)

- If computer slightly too slow: If can’t handle Update+Draw in one frame, can probably handle Update+Update+Draw in two frames
  - May look jerky but should play OK
- If computer way too slow (i.e. Update alone doesn’t fit in a single frame): we are doomed
- In both above cases, a clever program could see that GameTime.IsRunningSlowly == true and reduce level of detail
  - Most games don’t bother

When XNA gets behind (2)

• If particular frame took too long: call update extra times to catch up, then continue as normal
  – Player may notice slight glitch

• If paused in debugger: XNA will get way behind and give up, but will continue running OK when debugger resumed

“Heisenberg Uncertainty Principle”

• If you put in breakpoints, may notice Update being called more often than Draw, since the breakpoint makes you late

• Examining the timing of a system changes the timing!

XNA game loop: Variable Step

- Game.IsFixedTimeStep = false;
  - Update
  - Draw
  - Repeat
  - (more or less)

- Update should use elapsed time information