Navigating detailed worlds with a complex, physically driven locomotion: NPC Skateboarder AI in EA’s:

Mark Wesley
Senior Software Engineer
EA Black Box
What is skate?

• It’s a skateboarding game…

• A brand new franchise from EA Black Box, launched in September 2007 on Xbox 360 and PS3.

• Lots of fun
So what about the AI?

The Game Design called for:

• AI competitors in challenges (Races / Best Trick Contests / etc.)

• Ambient “Living World” AI skaters for atmosphere.

• (Note: There are also AI controlled pedestrians and vehicles, but that’s a separate topic.)
So where’s the problem?

- Detailed Collision Environment
- Skaters are fully physics driven (i.e. we can’t cheat)
- Fully physics driven skateboarding is quite a complex form of locomotion to steer.
- No ability to walk (until skate 2)
- Dynamic world (vehicles, pedestrians, other skaters, etc.)
Launch a trick incorrectly
(wrong place / direction / speed / time / …)
... then you’ll bail
Even curbs are hard...
Our Solution – AI Paths

• Record people playing the game
  – Records the “what and where”, not the “how”.
• Attach metadata to the path
• Save out as an XML file
• Process the paths into something usable in game.
Path Recording

- Store nodes at varying intervals whilst skating.
- Path Nodes consist primarily of:
  - Position
  - Velocity
  - Orientation
  - Width (calculated by terrain analysis)
Runtime Path Format

- Node elements are stored compressed, quantized and packed.
- Path is interpolated from nodes.
  - Position and velocity use a Hermite curve.
  - Orientations use slerped quaternions.
  - Nodes are stored whenever the interpolated data would differ from the original by more than a given threshold.
Recording A Path
Path Editor

• All in game, allows designers to:
  - View and select paths in the world
  - Trim, tweak and delete paths
  - Edit metadata (effectively allowing them to script who can use the paths and when)
  - Quickly iterate, test and debug.
Viewing A Path
Path Pre-Processing

- Detects branches etc.
- Available in-game for rapid WYSIWYG iteration by designers
- Also done offline to generate efficient binary data for the rest of the team (and the finished product)
Path Following

- Any skater can optionally have an AI Controller attached
- Looks at the skater’s current path
- Evaluates dynamic obstacles
- Looks at branches and other paths
- Tries to pick the best route
- Drives the skater with controller intents
AI Profiles

• Tunable by designers

• Influence ability, tricks performed and style of skating for each character

• AI skaters dynamically swap tricks when possible for variety.
It’s easy if there’s nothing in the way…
Dynamic Avoidance

- Evaluate all dynamic obstacles around the skater.
- No static analysis of the world (all we need is already in the path)
- Cut obstacles out of the paths
- Look for possible speeds that would allow skater to pass in front or behind other moving entities
Dynamic Avoidance: Ex 1
Dynamic Avoidance: Ex 1
Dynamic Avoidance: Ex 2
Dynamic Avoidance: Ex 2
Dynamic Avoidance: Ex 3
Added Bonus - Skitching
A Skitchable Obstacle
Within Path, Correct Direction
So Skitch It...
And Steer As Necessary
Skitching – Exit
Paths Diverge
Detatch from Vehicle
Narrow Path, Wide Object...
If Short Enough...
Run / Jump Over It
If Completely Blocked...
Give up and bail…
Respawn Just Past It…
Quick Tangent...

- All the above shots are from skate’s replay editor.
- All debug + diagnostics drawing goes through our replay system.
- Insanely helpful for debugging (for SEs and designers)
Automated Testing

• AI Controller could be attached to any skater.

• Therefore easy to make the game play itself.

• Useful for overnight soak tests etc.
Some Paths
Do we need many Paths?

- Large Open World
- Needed lots of good path data
- Our QA department helped out by generating a large amount of it for us.
Lots of Paths
Lots and Lots of Paths
Paths, paths, everywhere
Some Stats from *skate*

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of Paths</td>
<td>4,825</td>
</tr>
<tr>
<td>Total number of Nodes</td>
<td>250,747</td>
</tr>
<tr>
<td>Total number of Tricks</td>
<td>20,515</td>
</tr>
<tr>
<td>Total number of Branches</td>
<td>48,500</td>
</tr>
<tr>
<td>Total length of paths</td>
<td>465 Km</td>
</tr>
<tr>
<td>Total duration of paths</td>
<td>17.5 Hours</td>
</tr>
<tr>
<td>Total memory (if all paths were loaded simultaneously)</td>
<td>12.68 MB</td>
</tr>
</tbody>
</table>

- Note: 465 Km (290 Miles) is the equivalent of skating from Vancouver to Seattle and back, followed by 11 runs down Mount Everest.
Conclusion
Pros

• It worked
• Game Designers seem to like it
• Allows some scripting “for free” by simply constraining the paths used at a particular point
• A large path set combined with more random path constraints gives a nice emergent behavior
Review Praises AI?

"Playing solo in the career mode won't leave you feeling lonely. San Vanelona is somewhat of a haven for skaters; they flock there... You'll be doing a challenge and someone might cut in and skate your line. Or you'll be hunting for a new spot to skate and have P-Rod ride past you. These appearances are common, but not superficial. You can follow Rodriguez around town, which may lead you to a sweet spot that you didn't know about... (90%)"

(IGN Review, 2007)
Cons

• Skaters are constrained by the path network

• Requires a lot of data to be recorded

• Paths are invalidated if the underlying world moves.
Possible Extensions
Paths on Dynamic Objects
Paths on Dynamic Objects
Record the End User

• Record paths constantly, upload, process and share the data.

• Easy to generate an AI profile from already captured telemetry data.

• Asynchronous Online – AI equivalents of your friends appear around you!
Better use of NavMesh

• There’s Navmesh in the world already (for the pedestrians).

• Didn’t use it for skaters in Skate 1 (you couldn’t walk after all…)

• Originally planned to use it as a fallback in many cases in skate 2…
Smarter Path Usage

• Background process to weight paths based on object obstruction etc.

• Thinking multiple steps ahead (we don’t do any “path-finding”, because we’ve never actually needed to).
Questions?