Navigating Detailed Worlds with a Complex, Physically Driven Locomotion: NPC Skateboarder AI in EA's *skate*

Mark Wesley

EA Black Box
19th Floor, 250 Howe Street
Vancouver, BC, Canada V6C 3R8
mwesley@ea.com

The Author

Mark Wesley is a veteran video game programmer with 8 years experience developing commercial games across a variety of platforms. He's worked for major developers such as Criterion, Rockstar and EA Black Box, shipping numerous titles including Burnout, Battalion Wars, Max Payne 2, Manhunt 2 and *skate*. Although a generalist who has worked in almost every area of game programming from systems to rendering, his current focus is on gameplay and AI. Most recently, he was the skater AI lead on *skate* and is currently the gameplay lead on *skate 2*.

Introduction

This talk describes the motivation, design and implementation behind the AI for the NPC Skateboarders in *skate*.

The complexity of the physically driven locomotion used in *skate* means that, at any given point, there is an extremely large number of degrees of freedom in potential motion. In addition to this, the rules governing whether it is possible to navigate from any given point A to a secondary point B are entirely dependent on the skateboarder's state at point A. The state required at point A involves a large number of variables, as well as a complex set of previously executed maneuvers to have reached it.

Original Goal

The original AI design was extremely simple: Populate the world with NPC skaters having roughly the same abilities as the player. These NPC skaters would appear as "living world" ambient skaters to provide atmosphere during normal gameplay and also compete against the player during challenges.

The AI-based challenges fit into 4 main types:

- Races
- Point Scoring (by performing tricks)
- Follow-Me (skate a pre-determined route whilst the player follows)
- S.K.A.T.E. (setting and copying specific tricks or trick sequences)

Technical Design Considerations

1. Complex, exacting, physically-driven locomotion
2. No off-board locomotion (i.e. no ability to walk)
3. Detailed, high-fidelity collision environment
4. Static world combined with dynamic entities to avoid (pedestrians, vehicles and other skaters)
5. Short timeframe to implement the entire system
6. Experiences from the SSX team on their somewhat related solution

The combination of (1), (2) and (3) add a significant amount of complexity to even just the basics of successfully navigating from one point to another - without launching the correct trick, from exactly the correct position with the correct speed, orientation and trajectory, it is impossible to even do something as simple as jumping up a curb, let alone achieve some of the more complex maneuvers required in the game. This means that a standard waypoint or navmesh based system would simply not be sufficient as there are far too many dependencies on whether any given connection is actually navigable; let alone then expressing all of the additional information on how to successfully navigate that section.

Several team members had previously worked on the SSX franchise which solved a simpler but somewhat related problem with the use of player-recorded paths. Feedback about this system, both from those on the *skate* team and from those elsewhere in EA, was gathered and used to help design our solution.

The Chosen Solution – Player Recorded AI Paths

Given the considerations above, especially the time constraint, the somewhat tried and tested method of recording player paths was chosen. On top of this it was necessary to add various dynamic elements like branching, dynamic obstacle avoidance, trick-swapping and wipeouts, which would allow modification of the runtime behavior as required and add variety.

AI Path Recording

AI paths were generated by simply recording the player skating around the world. The paths recorded the "what and where" (e.g. position, orientation, speed and trick performed) rather than the "how" (e.g. controller inputs) so that we did not have to rely on determinism – the advantages being that it was then possible to diverge from a path as necessary at runtime for avoidance and it also coped with any minor changes in physics, animation and collision geometry during development.

AI Path Data Structure

Nodes were sampled at discrete points and each contained:

1. Position
2. Velocity (observed rather than internal velocity from physics) - used for both the Hermite curve approximation and for calculating the expected velocity at any point
3. Board orientation
4. Skater orientation
5. Time since the last placed node
6. Path width (how far the skater could stray to the left or right of this point and still be able to safely skate this path)
7. Some additional states, like whether the skater was crouching or if the board was flipped

Extended Node data – This was only present for all start-trick nodes, and at the start of all air trajectories. It did not exist for all other nodes (to save memory).

1. Trajectory (position and velocity)
2. Pitch (whether the skater performed a front or back flip)
3. Yaw (how many degrees the skater spins)
4. Trick (which trick was performed)

Each element was stored in a memory efficient manner (compressed vectors, compressed quaternions, quantized structures, bit-flags, etc.), and everything was packed together so that no memory was wasted on alignment.

In Game Recording

All recording functionality was available from a simple set of in-game menus. Existing debug rendering support was used to draw the paths as they were being recorded or edited.

Whilst recording a path, nodes were placed at various intervals behind the player. The decision of when to place the next node was determined by calculating the error between a high detailed recording of the player (with a node stored from every frame of the last few seconds) with the interpolated data that would be produced between this frame and the last node placed. As soon as the error at any frame exceeded some predefined thresholds then a node would be placed. On average, this resulted in one node being placed every 15 frames.

Rather than linear interpolation of the position, a Hermite curve was used (using the position and velocity at each node). Board and skater orientations were slerped, and the velocity was re-computed back from the positional Hermite curve rather than being interpolated.

Path widening was achieved as an automated off-line process where the collision geometry and surface gradient was evaluated on each side of the skater's recorded path to determine how far left and right the skater would be able to diverge from the path and still be able to skate sensibly.

Once created, AI paths could then be: loaded, edited, saved and deleted; all from within the game.

The path naming used a simple set of menus for selecting the exact path type. For ambient paths, it would generate the name as a GUID (so that multiple people could seamlessly work on ambient paths across the world). For challenge paths, it would automatically generate a path name using a standardized naming convention for the challenges.

Source Control

All path-editing operations (including path creation, renaming and deletion) would, on confirmation, save the relevant data onto the game console (in an intermediate XML format).

Committing these changes into our source control system was then achieved by running one small customized command-line tool which would automatically generate a change-list containing all relevant Add, Delete and Edit operations.

Path Pre-Processing

To minimize runtime processing, all relevant path branches were detected in an offline pre-processing step. This pre-processing step occurred in the tools pipeline, alongside

the steps which converted the intermediate XML AI Path data into its runtime binary representation. Paths for each challenge were packaged together, and ambient paths were injected into the world streaming data.

By default all builds of the game would load the binary path resources. Ambient paths were streamed in and out whilst skating around the world, and challenge paths were loaded as one path package as part of the challenge loading. However, XML path loading, combined with an exact equivalent of the offline pre-processing, was present in the non-final game builds to allow anyone (especially the challenge designers) to work with the intermediate data in an instant-feedback, zero-turnaround-time manner whenever required – this could be enabled via a startup.ini file, or toggled at runtime, and allowed the behavior of the ambient path streaming and the challenge loading to be toggled separately.

AI Path Following

The runtime path following primarily consisted of the following steps:

1. Evaluating where the skater was relative to their current path (also detecting if the skater had somehow become stuck).
2. Evaluating any nearby dynamic obstacles
3. Evaluating any path branches (and potentially changing their path)
4. Choosing the safest route to avoid obstacles
5. Choosing the controller intents necessary to achieve the desired goal (turn, push, crouch, brake, perform trick, etc.)
6. Passing the desired result data to physics (allowing the physics to constantly apply a very small blend to position, orientation and velocity to keep the skater on the intended route)

Branching

Path branching was achieved using the branch data built into the path during the pre-processing step. Path selection at branches (and path ends) was done using a heuristic which favored the least obstructed, easiest to get to path combined with a random factor to add variety. Scoring potential, compared to the current desired points, as well as trick potential, was also be evaluated in certain trick-based game modes.

AI Character Profiles

Each skater's profile was tuned by the game designers in *skate*'s existing attribute editor. Tunable elements included the skater's speed, trick weightings (for how likely they were to perform a specific trick), trick landing percentages (for how likely they were to bail a given trick), and the likelihood of them celebrating when they saw, or performed, an impressive trick or trick sequence.

Trick Swapping

Whenever a trick node was encountered during path following, the AI skater would determine whether to follow the originally recorded trick, or if any of the other tricks could be performed instead (based on a heuristic, using such elements as the trick's duration and whether it landed into another trick). The chosen trick would depend on the skater's profile along with their challenge objectives (e.g. they would always choose the required trick in a game of S.K.A.T.E. when copying the player's trick, whereas an ambient skater would pick a trick at random based on the weightings in their profile).

Wipeouts and Path Resetting

When landing, the AI would determine (based on their profile) if they should wipeout. In addition to these forced wipeouts, they would occasionally bail directly from collisions and physical constraints. Whenever the AI detected that a skater was stuck, and any attempts to un-stick them by analyzing potential alternative routes had failed, they were wiped-out as a last ditch "catch-all" solution.

Rubber Banding

During race and follow-me challenges, the AI skaters were subject to some subtle rubber banding to help with balancing. The rules of which were quite straightforward and all relevant values were designer-tunable:

1. Any AI Skaters that were ahead of the player were slowed down, the amount of slowdown blending smoothly from nothing, to full slowdown depending on how far ahead they were.
2. Any AI Skaters that were behind the player were sped up, the amount of speedup blending smoothly from nothing, to full speedup depending on how far behind they were.
3. Whenever the player was wiping-out, all AI skaters would slow down to their full slowdown speed.

Avoidance

For the pre-recorded paths to work correctly in a dynamic world, with pedestrians, vehicles and other skaters, it was necessary for dynamic avoidance to be combined with the basic path following.

The avoidance system worked by evaluating all relevant dynamic objects near to the player. Their estimated current

and future effect on the AI's path was determined, and potential solutions were evaluated.

When skating on a widened path section, the combination of all occluded segments of the path was subtracted from the navigable path data, and the center of the optimum navigable segment was chosen as the candidate destination (assuming that the path was not fully obstructed).

If no navigable segment was found (e.g. the skater was on a narrow path section), then possible speeds were evaluated for passing ahead or behind the set of obstacles. If avoiding a collision was not possible, then the skater would attempt to stop and/or potentially look for another path.

Simulating Out-Of-World Skaters

As only the world surrounding the player was loaded, it was entirely possible for an AI skater to be skating in part of the world that was not currently loaded. This was especially true in challenges that spread over a large area, like the race challenges.

To prevent the AI skaters falling out of the world in this case, whenever they were outside of the streamed-in world they had their normal physics disabled and were instead simply dragged along the AI path network at the desired speed (based on a combination of the recorded path speed and the skater's speed setting in their profile).

Ambient Skaters

For the most part, the ambient AI skaters simply used the existing skater AI support, but there were a few additional features:

- All ambient paths were streamed with the world (rather than being loaded by the relevant challenge).

- Ambient skater spawning was based simply on whether there was a currently loaded AI Path that started between the given minimum and maximum spawn distances around the player. Only 3 ambient skaters were ever allowed in the world at any time.

- Ambient skater un-spawning occurred whenever an ambient skater was further away from the player than a given un-spawn distance (this distance was further than the maximum spawn distance, to prevent them repeatedly spawning and un-spawning on the un-spawn boundary).

- On spawning, the model and character profile for each ambient skater was chosen at random from the currently loaded character models which were not currently being used for either skaters or other NPCs and excluding any characters which were still currently locked in the progression system.

- To add variety, the characters used by the ambient NPC skaters were rotated. Whenever 1 of the 3 possible AI skater models was not being used, it was optionally unloaded and swapped for a new model (depending on various elements such as how many times it has been used, and which models were currently usable).

To help populate the entire world with ambient paths, the challenge design team was briefly assisted by a number of *skate*'s QA testers – they helped to rapidly fill the world with the necessary AI path data once the world geometry was locked down.

Automated testing

Due to the architecture of the AI system, it was trivial to attach and remove an AI controller from any skater in the game. Attaching an AI controller to the player was used throughout development as a rapid means of testing and iterating on AI features within a minimal environment.

A simple automated stress / soak test was integrated into the game using the player-controllable AI which would make the player skate freely around the world using any available path data. Any blocking front-end screens were automatically skipped during this, and it was, therefore, trivially easy to leave any build running as if being played solidly all night. This provided a relatively realistic in-game soak test where systems such as the world streaming and physics were being exercised constantly.

Hooks for using the player-controllable AI were also provided to the automated testing team, with the vision of a specific set of paths and scripts being built to perform a completely automated play-through of the game.

Various functional tests were built alongside the development of the AI to prove that various features worked. These proved very useful not only for building the system and providing a reference implementation to any customers of the system, but also for catching when any part of the AI system was broken by another external system.

Statistics from the final version of *skate*

The following table shows some statistics from the AI path data used in the final version of *skate*:

Total number of Paths	4,825
Total number of Nodes	250,747
Total number of Tricks	20,515
Total number of Branches	48,500

Total length of paths	465 Km
Total duration of paths	17.5 Hours
Total memory (if all paths were loaded simultaneously)	12.68 MB

These were made up of 3,350 ambient (streamed) paths, and 1,475 challenge paths.

To put the total path length (465 Km) in perspective – that is equivalent to skating from Vancouver to Seattle and back, followed by 11 runs down Mount Everest.[1]

Note: Only the ambient paths in the streamed area around the player (generally between about 500 to 750 KB), along with the paths for the current active challenge (generally between about 50 and 200KB), were ever in memory at a given point, fitting within the 1 MB budget for the entire Skater AI system. This represented 0.2% of the available 512MB of memory on our target platforms.

Conclusion

Overall, the path recording system worked well. In addition to enabling the challenge designers to script specific behaviors for each challenge, it also provided a relatively simple way of collecting a large amount of general skating data for the ambient skaters to provide atmosphere during normal gameplay.

"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)

References

EA Canada. 2003. *SSX 3*. EA Sports BIG. (Video Game)

EA Canada. 2005. *SSX On Tour*. EA Sports BIG. (Video Game)

IGN. 2007. *skate Review*. Retrieved September 7, 2007, from the World Wide Web:
http://xbox360.ign.com/articles/818/818832p3.html

---

[1] Based on distance from Vancouver to Seattle being 185 Km, and height of Mt. Everest being 8.8 Km.