Animation-Driven Locomotion For Smoother Navigation

Bobby Anguelov
AI Programmer, IO Interactive

Gabriel Leblanc
AI Programmer, Eidos-Montréal

Shawn Harris
Senior Programmer, Big Huge Games
Format

- Introduction to Animation-Driven Locomotion (ADL)
- Big Huge Motion Planning
- Eidos / IO Interactive Motion Planning
- Hitman: Absolution Case Study
Animation-Driven Locomotion

- Animation defines spatial transformations
  - Movement can reflect kinematic properties of animations.
Motivations for ADL

- Invalid kinematics break immersion
- ADL systems allow complex motion sets
- Animators work directly reflected in game
Navigation and ADL

- What is the goal of using ADL for Agents?
  - Realistic, high fidelity visualization with complex continuous ranges of movement
  - Navigate within the constraints of the system.
Using ADL

- ADL is simple, usage patterns are hard!
  - Multiple approaches
  - Varying complexity
The Prototypes

- Slide & Blend
- Animation Driven
- Transitions
- Parametric Animations
- Parametric Transitions
- Step-Based Planning
- Learned Controllers
- Continuous Planning
The Prototypes

Complexity and Fidelity

- Slide & Blend
- Animation Driven
- Transitions
- Parametric Animations
- Parametric Transitions
- Step-Based Planning
- Learned Controllers

Continuous Planning
The Prototypes

ADL

- Slide & Blend
- Animation Driven
- Transitions
- Parametric Animations
- Parametric Transitions
- Step-Based Planning
- Learned Controllers

Continuous Planning
Common Thread

- Same Problem
- Multiple Solutions
- Choosing the best approach is difficult
The Prototypes

- Parametric Animations
- Parametric Transitions
- Continuous Planning
- Step-Based Planning
- Learned Controllers
- Slide & Blend
- Animation Driven
- Transitions
Reckoning’s ADL System

- ADL Roadmap
- Motion Planning
Reckoning ADL Roadmap–Player Movement

- Wanted ADL for player character
- Motion graph implemented
- Extracted data from animation node
Reckoning ADL Roadmap–Player Movement

- Orientation - Procedural and ADL
- Blending of movement information
Reckoning ADL Roadmap – NPCs

- Used tech developed for the player
- Attempted real time motion graph search
Reckoning ADL Roadmap – NPCs

- Used system defined for player character
Motion Planning – Developed Tech

- ADL
  - Linear and rotation data encoded
Motion Planning – Developed Tech

● Motion Graph
  ● Defined by designers
  ● Conditional Links
  ● Tree like interface
Motion Planning – Developed Tech

- Motion Graph
  - Became very large
  - Grouping for organization needed
Motion Planning – Facilities Available

- Animation Blending
- Additive Animation Blending
Motion Planning – Facilities Available

- Procedural ADL Adjustments
  - Ability to adjust linear and rotational output
Motion Planning

- How is motion planning completed?
  - Motion graph does all planning
  - Graph considers graph state, input, and game state.

- How do we ensure orientation?
  - Procedural orientation
  - Foot slide still occurs
Motion Planning

- How is navigation fidelity improved through this system?
  - Straight line movement looks great and foot slide is prevented
  - Overall increase in fidelity in movement and combat
Motion Planning-Detriment

- Fidelity to time cost high
- Foot sliding still occurs
Motion Planning - Benefits

- Low overhead for motion planning code
- Non-emergent results
- Easy addition of animation fidelity
- Good stepping stone
An ADL Approach to Foot Step Planning

- This system will:
  - Respect the kinematics of walking at all times
  - Reach destinations precisely with proper orientation
  - Use a per segment navigation approach without steering
The Prototypes

- Slide & Blend
- Animation Driven
- Transitions
- Parametric Animations
- Parametric Transitions
- Step-Based Planning
- Learned Controllers
- Continuous Planning
An ADL Approach to Foot Step Planning

- Why bother? Our needs:
  - Stay exactly on a path at all times
    - To support planned interactions during locomotion
    - Full body IK can only correct to a certain extent
  - Characters that are able to convey their mood through their navigation
    - Specific clips depending on context and turn angles
An ADL Approach to Footstep Planning

- This system is best suited for:
  - Characters with high constraints on their locomotion
  - Stable targets
  - Slow pace navigation
An ADL Approach to Footstep Planning

- Why slow pace navigation?
  - Longer strides delay reactivity
  - Runway requirements limit possible paths
  - ...probably just not worth it in fast moving clips
Approach Overview

- Requirements
- The footstep planner
- Dealing with foot sliding
- Collision avoidance
- Adding more emotions to locomotion
Animation requirements

- A complete animation set that covers what your character can achieve
  - Start, Move, Turn and Stop clips
- Detailed clips with information on current feet status
- Extracted information on translation and rotation
A typical animation set

- Start animations that covers all angles
  - $-180^\circ$, $-90^\circ$, $0^\circ$, $90^\circ$, $180^\circ$
A typical animation set (*turns*)
A typical animation set

- Give some slack to your animators!
- Animation clips can have different length as long as they reach critical points at the same percentage of the clip
- Failure to do so can result in bad blends
A typical animation set

- Stop animations for each leg, every angle
  - We choose this to eliminate NPCs stopping and then turning on themselves to reach final orientation
A typical animation set

- To build a complete navigation set for one stance:
  - 6 start clips
  - 8 turn clips
  - 1 move cycle
  - 10 stop clips
- At ~50 frames per clip, this requires more or less 1250 frames
- 410KB in our test bed after compression
- Double that if you need to support variable speeds
  - Not a must in this system
Clip details

- Need information on foot ground contacts, foot passing to properly branch to different clips
  - Annotations or analysis
- **Contact** will be more responsive but with less acting compared to **Passing**

Let your animators decide!
Footstep planning

- We do not want to start out stop clips unless we are exactly on the branching position.
- In ADL there are two critical spots where we need to be precisely on our foot branching position:
  - Before a pivot
  - Before our goal
  - Before interactions
Footstep planning

- Path needs to be stable for planning to work
Footstep planning

- Simple path containing two segments
- Plan will contain a **start**, a **turn**, a **stop** and several **steps**
- Need to stay on the funneled path at all times to prevent hitting obstacles (i.e. walls)
- First we need to turn exactly on the pivot
Footstep planning

- We know we’ll need at least our **Start** and the first part of the **Turn** clip, so let’s insert them in our plan
Footstep planning

- We can now insert **Steps**, one leg after the other, until we reach our **Turn** clip
Footstep planning

- We can now insert **Steps**, one leg after the other, until we reach our **Turn** clip
- As expected, we will almost never fit properly.
- We can either take the extra step and *reduce* the displacement of every clip or skip it and *augment* them
Footstep planning

- We want to select the plan that will minimize the modification to the displacements
- In this case, we will go with fewer steps but make them longer
- We want to spread the distance we were missing with 4 steps through the whole segment for uniformity
- Even so, we will introduce foot sliding
Footstep planning

- We repeat this process for every segment
- Calculate by how much you would overshoot or undershoot the current target and annotate the path with that information
Dealing with foot sliding

- With this type of planning the error will be at max half a footstep span
  - IK can usually hide this error well enough
- Path post processing is an elegant way to eliminate sliding altogether
Path Post-Processing: Funnel Algorithm

- Standard path-finding only cares about the shortest path
- Post-processing is used on navmesh paths for ‘smoothing’ usually something like simple funnel algorithm step
- Funneling results in paths hugging exterior navmesh edges so navmeshes are often eroded away from obstacle geometry
Path Post-Processing: Shortest Paths

- Why the obsession with the shortest path?
  - When path length variations of 10~15% won’t be noticed

- Why do we have paths that hug corners
  - When humans don’t walk that way

- Why do we try to hide an error that results from forcing animations onto our paths?

  **Why don’t we simply force our paths onto our animations**
Path Post-Processing: Corner Push Away

- We create a path push-away vector at each path vertex
- The push-away vector can be any vector directed away from the corner
- One approach is to average the orthogonal vectors of the previous and subsequent path segments at a vertex

Funneled Navmesh Path
Path Post-Processing: Corner Push Away

- We create a path push-away vector at each path vertex
- The push-away vector can be any vector directed away from the corner
- One approach is to average the orthogonal vectors of the previous and subsequent path segments at a vertex
- The path can be pushed away from corner and thereby its length can now be dynamically modified
Path Post-Processing: Reducing the Error

- The error per segment is always at most half a footstep
- Once a path is found we plan our footsteps and calculate the distance error per segment
- By pushing away from corners, we can increase the length of the two path segments at that vertex thereby decreasing the error for those segments
Path Post-Processing: Reducing the Error

- How much do we push per vertex?
- Multivariate optimization problem
- A simple approach can get us most of the way
- We can modify the push direction in favor of the larger error
- While there is an improvement in the error for both segments we keep pushing
Path Post-Processing: Reducing the Error

- How much do we push per vertex?
- Multivariate optimization problem
- A simple approach can get us most of the way
- We can modify the push direction in favor of the larger error
- While there is an improvement in the error for both segments we keep pushing
Path Post-Processing: Reducing the Error

- How much do we push per vertex?
- Multivariate optimization problem
- A simple approach can get us most of the way

- We can modify the push direction in favor of the larger error
- While there is an improvement in the error for both segments we keep pushing
Path Post-Processing: Reducing the Error

- How much do we push per vertex?
- Multivariate optimization problem
- A simple approach can get us most of the way

- We can modify the push direction in favor of the larger error
- While there is an improvement in the error for both segments we keep pushing
Path Post-Processing: Reducing the Error

- How much do we push per vertex?
- Multivariate optimization problem
- A simple approach can get us most of the way

- We can modify the push direction in favor of the larger error
- While there is an improvement in the error for both segments we keep pushing
Path Post-Processing: Reducing the Error

- How much do we push per vertex?
- Multivariate optimization problem
- A simple approach can get us most of the way

- We can modify the push direction in favor of the larger error
- While there is an improvement in the error for both segments we keep pushing
Path Post-Processing: Reducing the Error

- Path post-processing won’t be able to completely reduce all per segments errors in one pass
- The more processing time you give it the better the results
- ~10% error per segment is perfectly acceptable and can be hidden without causing foot sliding
Path Post-Processing: Aesthetics

- By pushing paths away from corners we can improve the visual quality of paths.
- Especially useful with regards to tight turns e.g. doorways and staircases.
- We can also prevent wall hugging while keeping the navmesh as close to obstacles as possible.
- Reduce number of path segments.
Using the footstep planner to our advantage

- ADL is all about empowering the animators
- The footstep planner allows customization of the locomotion as well as preparing for interactions
Using the footstep planner to our advantage

- Take this example plan containing two segments
- We can easily alter the plan by cutting a few steps and inserting a custom animation
Using the footstep planner to our advantage

- Take this example plan containing two segments
- We can easily alter the plan by cutting a few steps and inserting a custom animation
- Let's remove 3 steps (right - left - right ... )
Using the footstep planner to our advantage

- Take this example plan containing two segments
- We can easily alter the path by cutting a few steps and inserting a custom animation
- Let's remove 3 steps (left - right - left ...)
- We can replace it by an animation that will break repetition while ensuring we stay on path
Collision Avoidance in ADL

- Cant talk about locomotion without discussing collision avoidance
- Collision avoidance (CA) is tied to the locomotion system used in the game
- CA is a pretty common problem and so can be taken for granted
- There are certain considerations to take into account with ADL CA systems
ADL CA:

ADL has one major drawback when it comes to CA: **Latency**

- With most ADL systems, NPCs can only react/transition at footstep events / transition markers
- Reaction latency is the time between footstep events and varies across movement speeds
ADL CA: Latency

- Both the reaction latency and the resultant NPC motion need to be taken into account when doing ADL CA
- Therefore Collisions need to be **predicted** and not simply reacted to
- Reciprocal collision avoidance for pedestrians (RCAP) - 2010
- Latency was also a significant problem in the extremely dense 1200 character HM:A crowds, which made use of a simplified motion graph controller
Crowds in Hitman: Absolution – Kasper Fauerby

Wednesday, 15:30 - 16:30
Room 2006, West Hall, 2nd Floor
ADL CA: Lack of Steering

- Most CA techniques make use of velocity obstacles in some form or other – ORCA / RVO / Clearpath
- They also often tend to assume that characters are “steered”
- Steering allows for immediate and fine-grained manipulation of angular acceleration / speed
- Traditional steering often results in foot sliding
ADL CA: Lack of Steering

- Using velocity obstacles for ADL CA can be overkill
- Since we pre-plan motion, we know an NPC’s exact position and velocity at any given point in the path
- We simply can perform localized collision detection along NPCs’ planned paths
ADL CA: Footstep Re-planning

- Once a collision is detected, we can react to it by either changing speed or by navigating around it.
- With footstep planning all avoidance actions will require re-planning of all footsteps along the remaining path.
- This re-planning may also require a new path post process step.
ADL CA: Transitions

- Start / Stop transitions can be problematic since NPCs are effectively uncontrollable during the transition
- Transition animation trajectories should be kept as short as possible
- Non-linear motion during transition can make collision detection difficult
- Ensure that NPC’s are guaranteed to be collision free before triggering a start transition
Collision avoidance is a very game specific problem - there is no silver bullet

Hitman: Absolution also makes use of ADL although we don’t use foot step planning

Even so, HM:A is still a good case study for collision avoidance using pre-planned ADL motion
The Original Assassin is Back!

- The biggest Hitman game yet!
- Using the Glacier2™ Engine
CA in HM:A – Constraints Design

- In HM:A, we have a new gameplay mode called “Instinct” which allows agent 47 to predict all NPC paths.
CA in HM:A – Constraints Design

- In HM:A, we have a new gameplay mode called “Instinct” which allows agent 47 to predict all NPC paths

- Our NPCs **MUST** follow their paths exactly (on-rails)
- NPC paths need to look good and **NO** foot sliding
- Our post processed paths are converted to a set of quadratic Bezier curves
- We can’t constantly modify paths while performing CA
CA in HM:A – Constraints Locomotion

- We don’t use footstep planning
- NPCs don’t have turn animations – we blend in a banking animation when turning
- Discrete movement speeds – walk, jog, sprint, etc...
- In the end we built a really simple system to handle our specific avoidance needs
CA in HM:A – Collision Detection

- We make use of animation metadata to create a local collision horizon
- NPC motion is predicted for that collision horizon along the NPC’s path
- Collisions are detected by performing moving sphere intersection tests for the NPC’s predicted motion
CA in HM:A – Allowing Collisions & Conclusion

- There are always going to be cases where collisions are unavoidable. E.g. During start/stop transitions.
- In such cases, we simply allow the collision to occur and play an upper body act to try and hide the collision as much as possible.
Conclusion

- This system is not intended to replace all other approaches
- Pick what works best for your requirements
Pro’s

● High fidelity, continuous movement
● Customizable handling of turns
● Interactions can be planned
● Respects the path and destination at all times
Con’s

- Latency can make navigating a dynamic environment difficult
- High memory requirements
- Short distances aren’t friendly
Thanks to:
Alex Champandard – aigamedev.com
Michael Buttner – IO Interactive
Josh Watson – Big Huge Games
Matt Gray – Big Huge Games
Everyone at Eidos-Montréal
Questions?

bobbyan@ioi.dk
sharris@bighugegames.com
gabriel.leblanc@eidosmontreal.com