Self-Refining Games using Player Analytics

Matt Stanton et al.
SIGGRAPH 2014

Presented by YoungBin Kim
2014. 05. 30

Computer Graphics @ Korea University
Abstract

- Demonstrate technique in a prototype *self-refining game*
  - Dynamics improve with play
    - Ultimately providing realistically rendered
    - Rich fluid dynamics in real time on a mobile device
- Present a sampling approach
  - Concentrating precomputation around the states
    - That users are most likely to encounter
Introduction
Introduction

- Interactive simulation - Data-driven techniques
  - Trade precomputation time for runtime speed and detail
    - Enabling stunningly realistic animation
      - Curling smoke
      - Flowing cloth
      - Deforming bodies
  - Only as good as their precomputation
  - Dynamical spaces are so large
    - Cannot precompute everything
Introduction

• To address this challenge
  ▪ Developed a model *self-refining game*
    • Whose dynamics improve as more people play
    • Exhaustive precomputation is unnecessary
      ▪ User interactions are typically structured
      ▪ Explore only a vanishingly small subset of the configuration space
Introduction

- Model the game
  - As a state graph
    - Whose vertices are states
    - Whose edges are short transitions between states
    - At runtime
      - Control (phone tilt) determines which transition to follow
    - Some edges *blend* simulations
      - Returning to a previously computed state
Introduction

• Which states should we explore?
  ▪ Naive growth strategies
    • Construct vast state graphs
      ▪ Only barely overlap with states explored by real players
      ▪ Also contain significant visual errors
  ▪ Using player data
    • Enables a novel form of crowd-based sampling
      ▪ Concentrates on those states players actually visit
      ▪ Significantly better state graphs with far fewer visual artifacts
Introduction

• Present *self-refining games*
  ▪ Whose dynamics are discretized into a state graph
  ▪ Exploits a new sampling strategy incorporating real player data
    • Significantly outperform previous strategies
  ▪ Present an algorithm
    • STATERANK
      ▪ Estimates the global probability of each state
        • Relative to a player model
  ▪ Adapt this framework to free-surface fluids
    • Using a novel similarity metric and blending technique
Related Work
Related Work

- Crowdsourcing
  - Text recognition
    - [von Ahn et al. 2008]
  - Drawing classification
    - [Eitz et al. 2012]
  - Performing user studies
    - [Kittur et al. 2008]
- One important subgenre of this research studies games
  - Which intrinsically motivate players to perform tasks
    - Labeling images
      - [von Ahn and Dabbish 2004; von Ahn et al. 2006]
    - Designing biomolecules
      - [Cooper et al. 2010; Lee et al. 2014]
Related Work

- Crowdsourcing to games
  - Using player data to improve the gameplay experience
    - Build player-adaptive AI
      - [Houlette 2003]
    - Generate customized levels
      - [Zook et al. 2012]
    - Generate stories
      - [El-Nasr 2007; Thue et al. 2007]
Related Work

- Playback mechanism
  - Improve gameplay
    - By generating a continually-improving sampling
    - Then played back at runtime
  - Also reminiscent of the video-playback mechanics
    - *Dragon’s Lair* [Cinematronics 1983]
      - Although since we do not depend on human animators
        - Capable of generating vastly larger data sets

- Our game improvement method
  - Automatic and generalizes across a large class of games
Related Work

• Precomputed fluids
  ▪ Topic of extensive recent graphics research
  ▪ Modeling free-surface fluids remains a challenge
    • For data-driven simulation
      ▪ Due to the complexity

• We successfully model-reduce such liquids
  ▪ Using a state-tabulation approach
Related Work

• Explicitly tabulates arbitrary dynamics and rendering
  ▪ In an offline process
  ▪ Near-exhaustive Precomputation of Secondary Cloth Effects
    • [Kim et al. 2013]
    • Our graph structure and growth process are similar
      ▪ Adapt these ideas to liquids
        • Using a new similarity measure and blend function
Related Work

• Data-driven methods
  ▪ Only as good as their precomputation data
  ▪ We attempt to capture an entire space of trajectories
    • Through a continuous state sampling process
      ▪ Uses game analytics
      ▪ Focus on that subset of the dynamics that players really explore
  • Estimate state visit probabilities using an algorithm
    ▪ STATERANK
      • Computes the stationary distribution of a Markov chain
      • With transition probabilities derived from a player model
      • Similar to the PAGERANK algorithm
State Graphs
State Graphs

• *State graph*
  - Vertices are game states
  - Edges are transitions induced by player actions
  1. Initialize a new state graph
     • Use a heuristic player model
  2. Make the game available to players
  3. Collect traces of the paths
     • They take through this graph during play
  4. By repeatedly collecting player data
  5. Updating our player model
     • Using the updated model to grow the graph
State Graphs

• In a state graph
  ▪ Each edge
    • Associated with an animation
      ▪ Connecting its source and destination states
      ▪ Format of these animations is application-dependent
        • Any encoding of the dynamics that we wish to display
  ▪ Each vertex has N outgoing edges
    • Games that sample player input from a set of N discrete controls
  ▪ At runtime
    • System replays edge animations
      ▪ Determining which branch to take based on player control
State Graphs

- Initialize a new state graph
  - Begin at a start state
  - Simulate every possible outgoing transition
  - Continue this process
    - Generating $N$ new simulations from each state
      - Until we create a small $N$-ary tree
    - Eliminate by blending with interior edge transitions
      - Leading back to an internal node
      - Many of these blends were between dissimilar edges
        - Low quality

![Diagram of state graphs with edges labeled e1, e2, e3, e4, e5, e6 and e1, e2, e3a, e3b, e3c, e3d.]
State Graphs

- Improve the quality of the graph
  - By growing it using new simulation data
  - Grow the graph by replacing blend edges with simulation edges

- Growing the graph can be a continuous process
  - As long as we have space to store the results of new simulations
State Graphs

• Key challenge
  ▪ Determining which blend edges to replace

• Graph quality evaluation
  ▪ Quantify each edge’s contribution to the quality of the graph
  ▪ Simple strategy to reduce error
    • Greedily replace the blend edge most detrimental to the quality of the graph
  ▪ BASELINE
    • Worst-case quality measure
      \[ \max_{e \in B} (\text{err}(e)) \]
      ▪ \( B \) : set of blend edges in the graph
      ▪ \( \text{err} \) : application-defined estimate of a blend edge’s perceptual error
      ▪ Always replace the blend edge \( e_{\max} \) with the highest error
State Graphs

- Graph quality evaluation
  - In a simulation-based game
    - Game objective encourages players to pursue strategies
    - Players will never visit the vast majority of the state space
      - Rendering most of BASELINE’s additions to the graph wasteful
  - STATERANK
    - Measures the expected error
      \[ \sum_{e \in B} P(e) \text{err}(e) \]
      - \( P(e) \): probability of traversing the edge \( e \)
      - Replace the blend edge \( e_{\text{exp}} \) with maximum expected error
        \( P(e_{\text{exp}}) \text{err}(e_{\text{exp}}) \)
      - Infer \( P(e) \) from a player model \( P(c|v) \) giving conditional probabilities of controls \( c \) at each vertex \( v \)
Player Model
Player Model

• In this section
  ▪ Describe how we can learn player models from data
  ▪ Describe how we use these models to create self-refining games
  ▪ Use two different player models
    • Heuristic model to bootstrap the simulation
    • Learned model to guide our exploration
Bootstrap Model

• When the game is first created
  ▪ No player data exists

• To bootstrap state graph growth
  ▪ Use a heuristic player model $P_h(c|v)$
    • Which essentially guesses what players will do
    • Many heuristics are possible
      ▪ Best heuristic will vary by application
  ▪ Maintain the current control with probability $\alpha$
    • Otherwise choose an alternate control uniformly at random

$$P_h(c|v) = \begin{cases} 
\alpha & \text{if } c = c_v \\
(1 - \alpha)/N & \text{otherwise}.
\end{cases}$$

• Combining this heuristic player model with STATERANK
  • SR-HEURISTIC
• Learn our player model
  ▪ From traces of player traversals of the state graph
    • Each trace consisting of a list of vertices visited
      ▪ And the control selected at each vertex
  ▪ $P_{obs}(c|\nu)$
    • Be the observed conditional control probabilities
      ▪ Computed by normalizing control counts at $\nu$
  ▪ $P_{obs}(\nu)$
    • Be the observed probability of visiting $\nu$
      ▪ Obtained by normalizing the number of visits to $\nu$ by the total number of vertex visits
• To generalize our model to unvisited states
  • Assume that players will take similar actions in states
Player Analytics Model

- Learn our player model
  - Implement our player model
  - Using a kernel density estimator
    - Combined with a Markov prior with weight $\epsilon$
      \[
P(c|v) \propto \sum_{u \in V} \sum_{c_u = c_v} w_u P_{obs}(c|u) P_{obs}(u)
      \]
      \[
w_u = k_{tri}(r, pdist(u,v)) + \epsilon,
      \]
      - $k_{tri}(r, x)$: triangular kernel with radius $r$
      - $c_u$ and $c_v$: controls of the simulation clips generating $u$ and $v$
      - $V$: set of vertices in the graph
      - $pdist$: inexpensive distance function
    - Combining this player model with STATERANK
      - SR-CROWD
Application to Liquids
Application to Liquids

• Our liquid simulations
  ▪ Using PCISPH [Solenthaler and Pajarola 2009]
    • Graph vertices $v$
      ▪ As lists of liquid particle positions and velocities
    • Graph edges
      ▪ As sequences of signed distance functions $e = [∅^1, ..., ∅^k]$
        • $k$-frame animations
        • $k = 10$, transitions of 1/3 of a second
  ▪ Video rendered
    • Using Mitsuba
Application to Liquids

• $dist(e_i, e_j, c)$
  - For blending
  - Based on energy and detailed liquid shape information
• $pdist(e_i, e_j)$
  - For player model
  - Compares only coarse shape descriptors
• $blend(e_i, e_j)$
  - Clip blending function
Edge Distance

- Perceptually-motivated error function
  - Incorporating both about the liquid’s shape and its energy

\[
dist(e_i, e_j, c) = \text{norm}_e(e_i, e_j) \left( \text{dist}_s(e_i, e_j) + \right.
\]
\[
\left. w_e \text{dist}_e (e_i, e_j, c) \right).
\]

- \( \text{dist}_s \): distance attributable to the shape of the two states
- \( \text{dist}_e \): distance attributable to the energies of the two states
- \( \text{norm}_e \): normalization term
- \( w_e \): controls the relative priority of the shape and energy terms
  - set \( w_e \) so that for edges \( r_i \) and \( r_j \) where the fluid is nearly at rest

\[
\text{dist}_s (r_i, r_j) \approx w_e \text{dist}_e (r_i, r_j, c)
\]
Edge Distance  
- Shape distance

- $\textit{dist}_s$ metric  
  - Penalizes the blending of animations  
    - Which contain liquid in very different shapes

$$\text{dist}_s(e_i, e_j) = \sum_{f=1}^{k} \text{vol}(\phi_i^f \triangle \phi_j^f)$$

$$X \triangle Y = X \cup Y \setminus X \cap Y$$
Edge Distance
- Energy distance

- $dist_e$ metric
  - Penalizes the blending of animations
    - That have very different energies

\[ E(v, c) = T(v) + V(v, c) \]

- $T$: kinetic energy
- $V$: potential energy
- $c$: incoming control

\[ dist_e(e_i, e_j, c) = \gamma |E(v_i, c) - E(v_j, c)| \]

\[ \gamma = \begin{cases} 
  c_{\text{gain}} & \text{if } |E(v_i, c) - E(v_j, c)| < T_0 \\
  c_{\text{loss}} & \text{if } |E(v_i, c) - E(v_j, c)| \geq T_0 
\end{cases} \]

- $v_i$ and $v_j$: destination vertices (final frames) of $e_i$ and $e_j$
- $T_0$: approximately the residual kinetic energy of the fluid when it is visually at rest
Edge Distance
- Energy normalization

- $\text{norm}_e$
  - Normalize the previous two terms

\[
\text{norm}_e(e_i, e_j) = \begin{cases} 
0 & \text{if } T_{\text{avg}} < T_0 \text{ and } c_i = c_j \\
\frac{1}{\sqrt{T_{\text{avg}} + T_0}} & \text{otherwise.}
\end{cases}
\]

\[
T_{\text{avg}} = \frac{1}{2} (T(v_i) + T(v_j))
\]
Player Model Distance

• STATERANK requires
  ▪ Perform neighbor searches
    • Using a brute-force scan of all vertices in the graph
      ▪ Function we use to compute vertex distances must be fast
      ▪ Therefore use a more efficient coarse shape similarity function pdist

• Compute a shape descriptor $d_i$
  ▪ For each edge $e_i$
    • By dividing the fluid domain into a 6 x 6 x 6 grid
    • Computing the average fraction of each cell
      ▪ That is occupied by liquid

$$\text{pdist}(e_i, e_j) = \|d_i - d_j\|_2$$
Blending

- Construct animations for blend edges
  - By blending signed distance functions
  - Using convex combinations of three signed distance functions
    - The source $\emptyset_s$
    - The destination $\emptyset_d$
    - The union of their shapes $\min(\emptyset_s, \emptyset_d)$

$$\text{blend}(\phi_s, \phi_d, t) = w_s \phi_s + w_d \phi_d + w_{s\cup d} \min(\phi_s, \phi_d)$$

$$w_s = \text{clip} \left( \frac{(1 + \ell - 2t)}{(1 + \ell)} \right)$$

$$w_d = \text{clip} \left( \frac{(2t + \ell - 1)}{(1 + \ell)} \right)$$

$$w_{s\cup d} = 1 - w_s - w_d$$

- $0 \leq t \leq 1$: denote the position in the blend
- clip: clips its argument to lie between 0 and 1
- $\ell$: parameter that limits the blending coefficient applied to the union ($= 0.1$)
Implementation

• Constructed a distributed simulation system
  ▪ Carry out the large-scale state graph explorations
  ▪ Worker nodes
    • Perform simulation and render animations
  ▪ Master node
    • Orchestrates computation by maintaining the graph structure
    • Computing edge priorities and distances
    • Assigning work to the workers
• Deploy this system
  • On Amazon EC2
    ▪ Configurations featuring up to 40 worker nodes
• System performed over 8600 CPU-hours of computation
  • Generated over 1.6 TB of data
Optimizations

• Lazy relinking
  ▪ Does not attempt “relinking” of existing blend edges
    • When new edges are created
• Pre-publish relinking
  ▪ Before playing a graph we attempt to relink every blend edge
• Animation caching
  ▪ Accelerate distance computations
    • Needed for nearest neighbor search
    • By caching voxel occupancy information in memory
      ▪ Our implementation uses memcached
• Energy pre-filtering
  ▪ Only perform full distance evaluation
    • On the $k$-closest graph edges ($k = 100$)
Mobile Client

- Android client application
  - The key feature
    - Ability to continuously play back short (1/3 second) videos
      - Without lag between them
  - When a player selects a game
    - Client downloads the most recent version of the game’s state graph
    - Then downloads and caches any videos for edges in the current graph
  - Use device accelerometer data to select game controls
  - After each play session
    - Client uploads a list of visited graph vertices
      - And the control selected on each visit to our server
Evaluation
Evaluation

- Grew state graphs for our fluid game
  - Using three different graph error measures
  - SR-HEURISTIC and SR-CROWD
    - Growth using STATERANK
      - Either a heuristic or crowdsourced player model
  - BASELINE
    - Only prioritizes growth
      - Using local conditional control probabilities
        \[ P_h(c|v) = \begin{cases} 
          \alpha & \text{if } c = c_v \\
          (1 - \alpha) / N & \text{otherwise.} 
        \end{cases} \]
        \[ \alpha = 0.8 \]
- Fluid simulations
  - 42K-particle PCISPH simulations
Evaluation

• Grew each of our graphs
  ▪ Until graph size reached 200K frames
    • Approximately 4,300 CPU-hours were used, per graph
      ▪ To compute each graph’s 1.8 hours of animation
  ▪ Paused graph expansion
    • At 10K, 20K, 50K, 100K, and 200K frames
    • So that the graphs could be played by a group of six test players
      ▪ Yielding gameplay traces for all graphs at these checkpoints
Evaluation

- 200K-frame state graphs
Evaluation

- Predicting Player Behavior
  - Edges visited at least 10% -> Red
  - Edges visited at least 2-10% -> Orange
  - All others -> Gray
Evaluation

- Observed Error During Game Play

- Observed Time Playing High-Error Animations
Evaluation

- Observed Error During Game Play

Observed Error During Game Play

“high error” threshold

Percentage of Play

Simulation Error

0.0001 0.001

BASELINE

SR-CROWD
Limitations
Limitations

• Total dynamic complexity
  ▪ Even simple generalizations of the dynamics
    • Would overwhelm our system
• Limits of control
  ▪ Increasing the temporal or spatial control resolution
    • Explodes the state space
    • Causes technical problems
• Range of applicable phenomena
  ▪ Some phenomena might be less forgiving
• Single-viewpoint rendering
• Storage requirements
• Applicability to existing games
Conclusion
Conclusion

• *self-refining games*
  - Dynamics continuously improve based on player analytics
    - Player data can be successfully exploited
      - To capture very complex dynamical systems
  - Player-driven state sampling
    - Enables us to deliver high quality rendered content in realtime

• Further research
  - Precomputed dynamics models with other virtual elements
  - Create more flexible models
  - Address other limitations
  - Create compelling and immersive virtual worlds