Implementing a Rewindable Instant Replay System for Temporal Debugging

Mark Wesley
Lead Gameplay Programmer – 2K Marin

GAME DEVELOPERS CONFERENCE® 2013
SAN FRANCISCO, CA
MARCH 25-29, 2013
EXPO DATES: MARCH 27-29

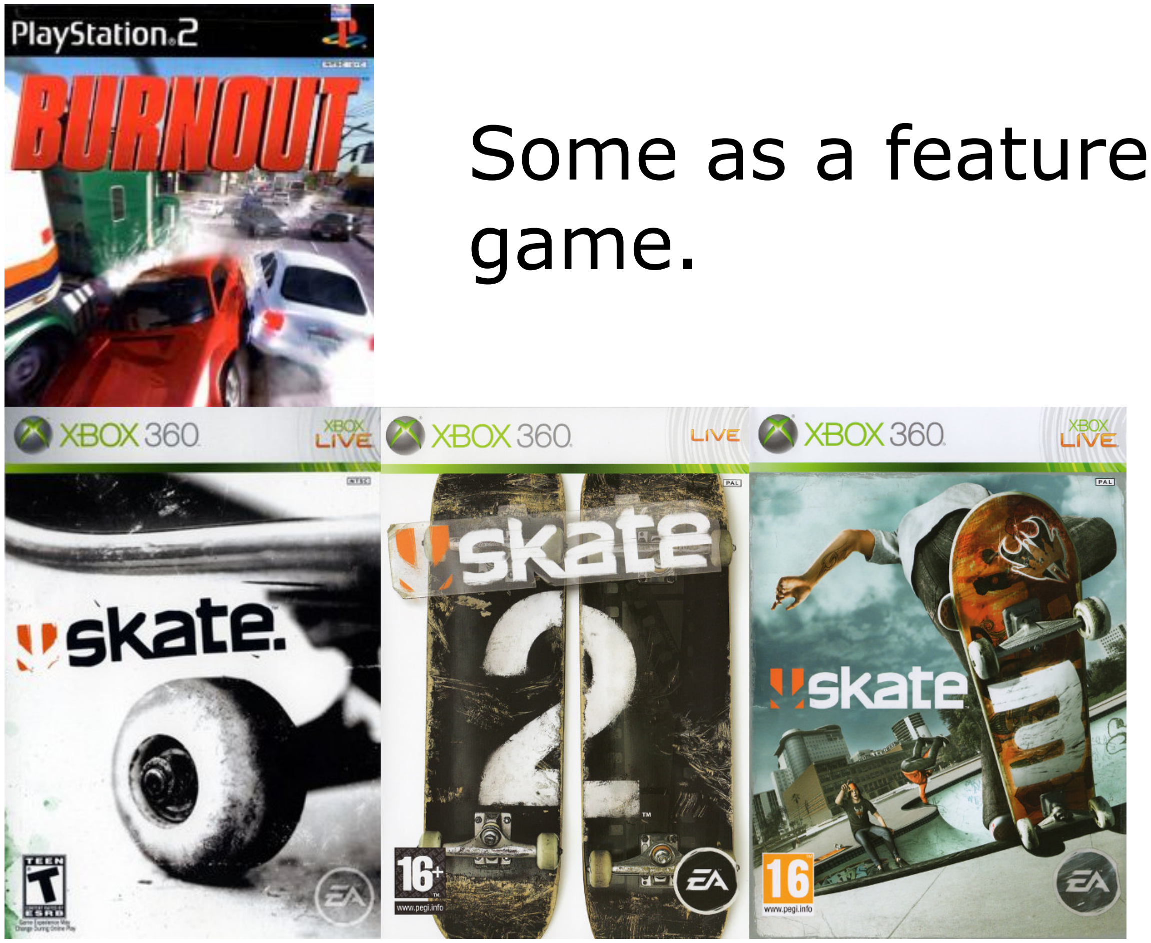
Talk contents:

- What is a “Rewindable Replay” system?
- How to implement one for debugging
- How to use it
- How to make it shippable
- Questions

PlayStation 2
BURNOUT
XBOX
REIGN OF FIRE
Let The Battle Ignite
NINTENDO GAMECUBE
BATTALION WARS
XBOX
MAX PAYNE 2
THE FALL OF MAX PAYNE
PlayStation 2
manhunt 2
XBOX 360
skate.
XBOX 360
skate 2
XBOX 360
skate 3
XBOX 360
NEED FOR SPEED THE RUN
LIMITED EDITION
EDICIÓN LIMITADA
XBOX 360
BIOSHOCK INFINITE

Several of those included Replay of some form

Mysterious Current Project



Some as a feature in the final game.

- Burnout:
  - Rewindable Crash Replays
  - Deterministic Race Replays
- Skate Trilogy:
  - Rewindable Video Replays

Some as a non-shipping feature added purely to aid development.

Mysterious Current Project

What is Rewindable Replay

- It's Rewindable…
- It is **NOT** determinstic Replay
    - Requires no determinism, therefore:
        - It is extremely robust
        - It is far more useful

What is Rewindable Replay

- Video-style replay
  - As seen in many sports games
- Like a VCR/PVR attached to your game.
- **BUT** you can:
  - Move the camera around
  - Enable debug info

If a picture paints a 1000 words…

- Then how about a video?
- Note:
  - I cannot show our game at this time
  - Ported code back to Unreal 3 in 1 hour
- http://youtu.be/x1rgEtC3bTc

Sounds expensive to make?

- To make it shippable, yes
- But for a dev only version, no
    - For internal use only
    - Doesn’t have to be pretty
    - Easier to find some spare memory

Development time

- Basics can be done in 2 days
  - Debug Draw and Persistent Entities
- A complete version in just 1-2 weeks
  - Skeletal meshes, Temporary Entities
- Will pay for itself very quickly

Implementation (Recording)

- Circular Buffer of frames
    - (e.g. last 900 frames =30s @ 30fps)
- Store all debug draw
    - In minimal form
        - E.g. As a Sphere or AABB not as lots of lines
- Store data on relevant game entities

Minimize Memory Usage

- Use compressed / compacted structures:
  - Store 3D vectors as 3 floats (not SIMD Vec4s)
  - Store orientations as compressed quaternions
  - Store debug text in 8 bit (not wide) chars
  - Store bools as bitfields / bitmasks
  - Pack everything together
  - A “Bit Stream” read/writer can be handy

Storing Debug Draw

- Merge nearby 3D text into “paragraphs”
- Clamp max debug draw per frame
  - But flag the frame as being overbudget
- Keep large static stuff out of replay
  - E.g. Navmesh, Collision meshes, etc.
  - Just draw them live

Storing Entities

- For relevant entities:
  - Need a fast way of iterating over these
  - Store:
    - A unique ID (e.g. { Ptr, SpawnTime })
    - 3D Transform
    - Bones for skeletal meshes (optional)
    - Any other important display info (e.g "Damage")

Store Game Camera

- In-game camera transform
  - Useful for debugging camera
  - Shows game from user's perspective
- Allow use of manual cam too

Implementation (Playback)

- Pause the game simulation
- For the current frame:
  - Render that frame’s debug draw
  - Use the stored transforms for each entity
- Allow scrubbing / rewind / fast forward

Implementation (Playback)

- Entities (re-use those in paused sim):
  - Render using data from the replay
  - For sub-frame blending:
    - Find matching entity in neighboring replay frame
    - Interpolate data as appropriate
      - Slerp quaternions
      - Lerp vectors

Short Entity Lifetimes (1)

- Entity didn't exist at 1st frame of replay?
  - Was spawned during replay
  - Exists at end (so in current simulation)
  - If not stored in current frame - don't draw it

Short Entity Lifetimes (2)

- Entity unspawned before end of replay:
  - There’s no entity to reuse…
    - Show a debug draw representation
    - Re-create entity (using the same mesh)
      - Harder if you already unloaded the assets…
        - Re-use a similar asset?

Replay – what is it good for?

- Combine with good in-game diagnostics
  - E.g. Lots of Debug Draw
- Temporal Debugging
  - Anything that occurred in the past
  - Understanding how things change over time

Replay – what is it good for?

- Temporal Debugging examples:
  - AI decision making, pathfinding, etc.
  - Physics collisions / reactions
  - Gameplay actions
  - Animation
  - Multiplayer replication

Vs. Traditional debugging tools

- Visual Studio etc.
  - Good for low-level debugging…
  - Terrible for higher-level debugging
    - Slow to get a bigger picture view
    - Only show the "now"
    - Very unfriendly for non-programmers
    - Particularly poor with optimized code

Vs. Traditional debugging tools

- Log Files / TTY
    - Provide a partial history of what happened
    - Hard to:
        - Parse hundreds of pages of these
        - Visualize game state in 3D
        - Associate with in-game

Gameplay debugging advice

- Use higher level tools / viewers!
- E.g. a Video Replay System.

Debugging Example

- http://youtu.be/jUpTGeszM4o

Memory usage

- Anything > 1MB can give a “useful” replay
- To find more memory:
  - On PC you probably have plenty.
  - On Consoles
    - Use the extra devkit memory…
    - Only store a small buffer, stream the rest to disk or over the network to your PC.

If you're still short of memory…

- Store less stuff…
- Compress / Quantize data
- Store only every N frames
  - Can be variable, and on a per entity basis
    - Does add a lot of complexity…

Useful Extensions

- Debug draw "channels"
  - E.g. AI, Physics, Animation
  - Always stored, but only displayed as required
- Serialize entire entity
  - Less reliant on debug draw
  - Can allow for lower-level debugging too

Useful Extensions Continued…

- Save / Load replays
  - Attach to bug reports etc.
  - Great for hard/slow to repro issues
- Export replays live to an external viewer
  - Provide a better GUI outside of game

If you wanted to ship it…

- Someone will always request this…
- It is quite a lot more work
  - Replaying everything (particles + audio)
  - Fitting in retail memory
  - Robust manual camera (with collision)
  - Nice GUI

If you still wanted to ship it…

- Design it in from the start
  - Use stateless parametric particle effects
  - Plan for the memory overhead
  - Plan for how you spawn / unspawn entities
  - Plan for how you stream assets
  - Plan for how you submit your render data

Replaying Particles

- If you don’t have a parametric system…
- Fake it by:
  - Storing start and end markers
  - Playing effect forwards by abs(deltaTime)
- http://youtu.be/bzwdPoKP80k

Any Questions?

Slides and contact information available at:

www.markwesley.com