# Kahawai: High-Quality Mobile Gaming Using GPU Offload

*Authors: Eduardo Cuervo, Alec Wolman, Landon P. Cox, Kiron Lebeck, Ali Razeen, Stefan Saroiu and Madanlal Musuvathi*

*Conference: MobiSys 2015*

# Motivation

- Mobile devices (tablets, smartphones and laptops) have less powerful GPUs than gaming consoles and high-end desktops because of power consumption limitations
- Mobile devices will have less powerful GPUs in the near future because of limited improvements in battery technology and form factor limitations
- Problem: Mobile devices cannot produce game visuals of the same quality as gaming consoles and high-end desktops because they have less powerful GPUs
- Solution: Offload GPU computation from mobile devices to servers to obtain high quality game visuals

# Thin Client

- Thin client approach is currently used in industry (Playstation Now and Nvidia Shield)
- Key idea: Server (with powerful GPU) executes the game and renders the game visuals and audio, and sends the rendered output to the mobile device
- Thin client has two weaknesses: transmitting high quality visuals requires high bandwidth and offline gaming is not supported

# Collaborative Rendering

- Paper proposes collaborative rendering to decrease required bandwidth for cloud gaming and support offline gaming
- Core idea 1: Less bandwidth is required if server and mobile device work together to render the game visuals
- Core idea 2: Method of collaboration is mobile device produces low-fidelity (low quality or less frames) visuals and works with server to obtain high-fidelity (high quality or more frames) visuals
- Two techniques for collaborative rendering: delta encoding and client-side I-frame rendering
- Requires that two concurrently executing game instances produce the same visuals

# Delta Encoding

- Key idea: High and low quality frames of most games share the same visual structure (majority of visual information)
- Typically, the compressed difference between high and low quality frames will contain significantly fewer bits than the compressed high quality frame
- Requires a way to generate high detail and low detail versions of frames (supported by many desktop PC games)
- Server concurrently renders high quality and low quality frames with the low quality frames matching the frames rendered by the mobile device

# Delta Encoding

- Observe that both low and high quality frames share the same visual structure
- The only difference is that the high quality frame contains fine-grained details


Low quality frame


High quality frame

# Delta Encoding

- Step 1: Mobile device sends input to server
- Step 2: Mobile device renders low quality frames while server renders both high quality and low quality frames
- Step 3: For each visual, the server computes a delta frame. The delta frame is the pixel-by-pixel difference between the high quality and low quality frames
- Step 4: Server compresses (i.e. encodes) the delta frames and sends them to mobile device
- Step 5: Mobile device decompresses (i.e. decodes) the delta frames and applies them to the corresponding low quality frames to obtain high quality frames

# Client-side I-frame Rendering

- Key idea: Take advantage of the way video is structured to reduce data sent by server to mobile device
- Mobile device renders high quality frames at low rate while server renders high quality frames at high rate
- Requires that mobile device be able to render high quality frames at fast enough rate (at least 6 frames per second)
- Requires access to game engine source code
- In particular, need access to game engine source code to prevent mobile device from rendering P-frames

# Client-side I-frame Rendering

- Compressed video is composed of three types of frames: I-frames, P-frames and B-frames
- I-frames are large in size and self-contained. An I-frame contains all the information needed to display the visual
- P-frames are relatively small in size and reference prior frames. A P-frame and all the other frames it references are needed to display the visual
- Ignore B-frames (not needed to understand technique)

# Client-side I-frame Rendering

- Step 1: Mobile device sends input to server
- Step 2: Mobile device renders only I-frames while server renders both I-frames and P-frames in the video. Only high quality frames are rendered
- Step 3: Mobile device encodes I-frames
- Step 4: Server compresses (i.e. encodes) the video, replaces the I-frames with empty frames and sends the video to mobile device
- Step 5: Mobile device merges its encoded I-frames with the video received and then decompresses (i.e. decodes) the result to obtain high quality frames

# Input Handling

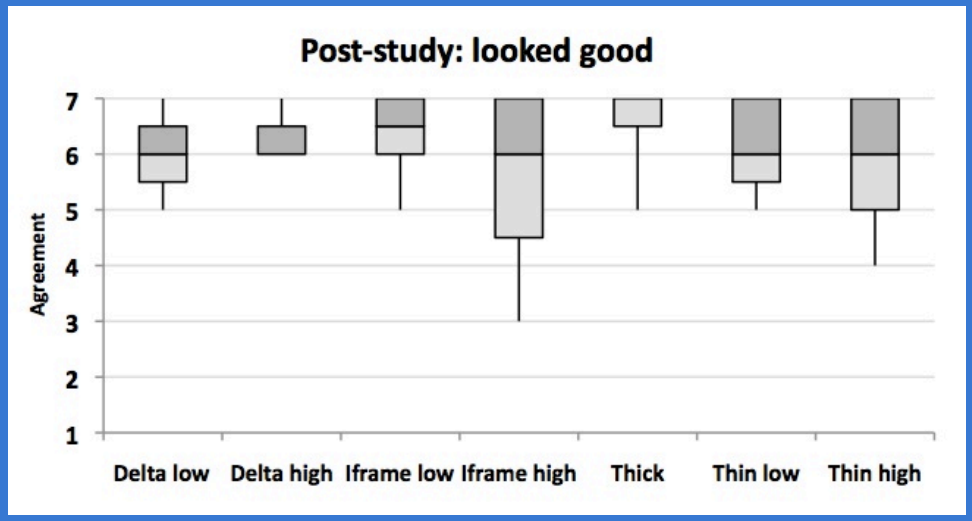
- The mobile device time-stamps inputs with the appropriate frame numbers before sending them to server
- The time-stamping and asynchronous input processing ensure that the game instances running on the mobile device and the server remain in sync
- Pipelining and adaptive clocking are used to reduce input-to-output latency
- Pipelining separates the major tasks in Kahawai into 7 asynchronous stages that are run in separate threads and each stage depends on the output of the previous stage
- Adaptive clocking ensures inputs are sampled at a decent rate to avoid significant decreases in pipeline throughput and significant increases in input-to-output latency

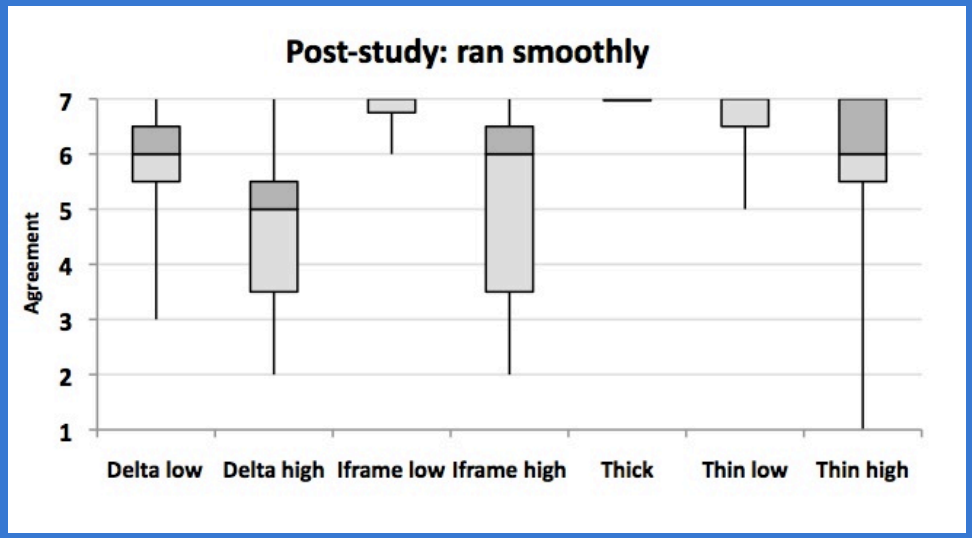
# Experimental Evaluation Part 1

- Question 1: How does collaborative rendering affect the gaming experience of users compared with thin client?
- Experiment: User study with 50 people
- The users play a portion of a level on Doom 3 and complete a post-study survey that assesses their game-play experience
- Users are asked how strongly they agree with the statements ‘the game looked good’ and ‘the game ran smoothly’ on a 1-7 scale
- 1 represents strong disagreement
- 7 represents strong agreement

# User Study Results and Analysis

- Perception of visual quality is roughly the same for both collaborative rendering and thin client
- Perception of game smoothness is roughly the same for both collaborative rendering and thin client
- User experience is roughly the same for both collaborative rendering and thin client

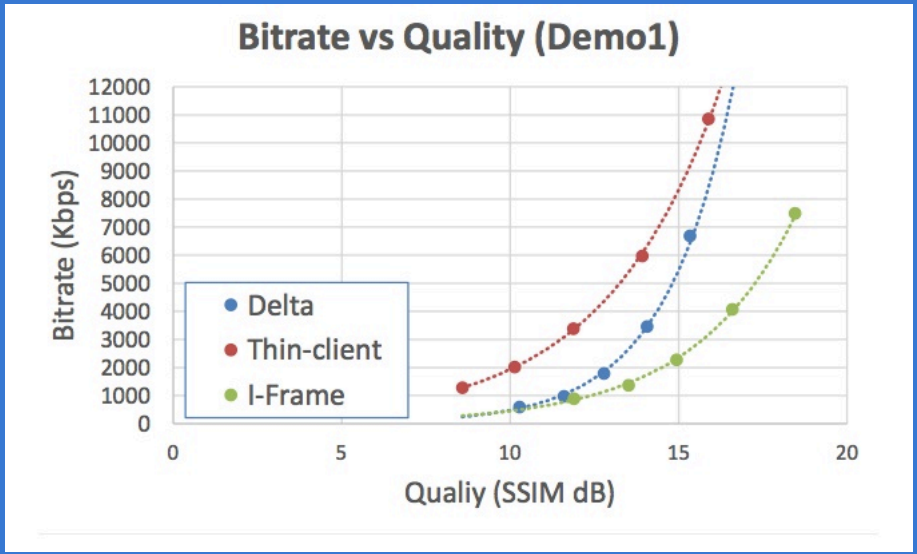


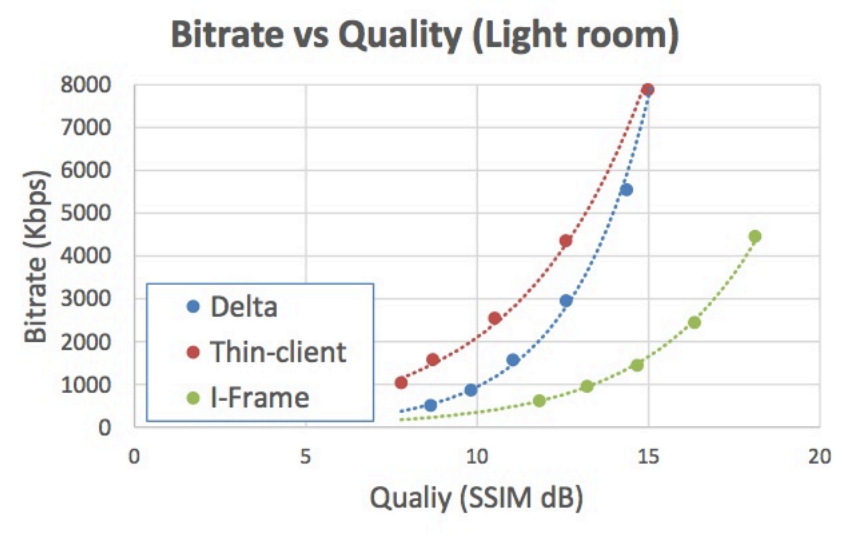



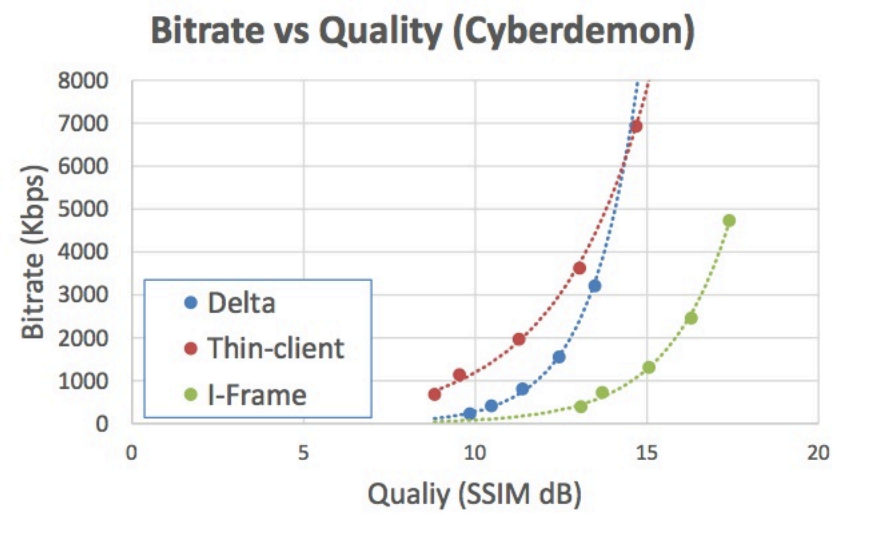
# Experimental Evaluation Part 2

- Question 2: For each of Kahawai's collaborative rendering techniques, how do the bandwidth requirements and visual quality compare to those of thin client?
- Experiment: Bitrate versus quality
- For 3 different scenes in Doom 3, vary the compression factor and plot the bitrate (i.e. bandwidth) used to transmit the frames against the quality of the frames generated
- Compare delta-encoding, client-side I-frame rendering and thin client approaches
- Image quality is measured using SSIM, which shows how similar a frame is to the highest quality version of that frame
- Horizontal axes in graphs show SSIM (dB) logarithmic scale

# Bitrate versus Quality Results



# Bitrate versus Quality Analysis

- Delta encoding provides better quality visuals than thin client with low bandwidth (less than 0.6 Mbps)
- Thin client needs up to 6 times as much bandwidth as client-side I-frame rendering to achieve high quality visuals (at least 15.23 on quality scale in graphs)

# Related Work

- MAUI and CloneCloud perform automatic code offload from mobile device to server
- MAUI and CloneCloud focus on CPU computation offload while Kahawai focuses on GPU computation offload
- Outatime is a cloud gaming system designed to mask network latency. The server predicts inputs, renders speculative frames of possible outcomes, and sends them to mobile device in advance
- Kahawai substantially reduces bandwidth required for cloud gaming at cost of moderate increase in input latency while Outatime masks substantial network latency at cost of extra bandwidth

# Future Work

- Combine both delta encoding and client-side I-frame rendering into one collaborative rendering technique to get more bandwidth savings
- Integrate Outatime and Kahawai into low-bandwidth GPU offload system that is resilient against high network latency

# Strengths

- When the mobile device is connected to the server, collaborative rendering produces high quality game visuals using significantly less bandwidth than thin client
- Mobile devices can run games offline but with low quality visuals
- To enable Kahawai for all games built on top of a particular game engine, only minimal changes need to be made to the game engine source code
- The use of pipelining and adaptive clocking to reduce input-to-output latency

# Weaknesses

- Kahawai does not work smoothly enough with high network latency
- Delta encoding can require more bandwidth than thin client when more objects are present in high quality scene versions and/or low quality scene versions contain more randomness than high quality scene versions
- Two concurrently executing game instances must produce the same graphical output
- Without game engine source code, Kahawai may not be able to be used in games built on top of this game engine (in particular, cannot use client-side I-frame rendering)