Conduit

Efficient Video Compression for Live VR Streaming

Gregory Rose
Aakash Patel

Live VR streaming

Live VR streaming:

Livestreaming 360° panoramic 4K 3D video from a panoramic camera to a client's HMD.

Many events to livestream in VR

Problem: VR needs a lot of bandwidth

- 3D 4K video: **20 Mbps**
- US Average Internet: **11 Mbps**
- 4G: **8 Mbps**

Source: Akamai

Solution: View optimization

View optimization: **optimize**, or compress the video for user's **view** (what they can see)

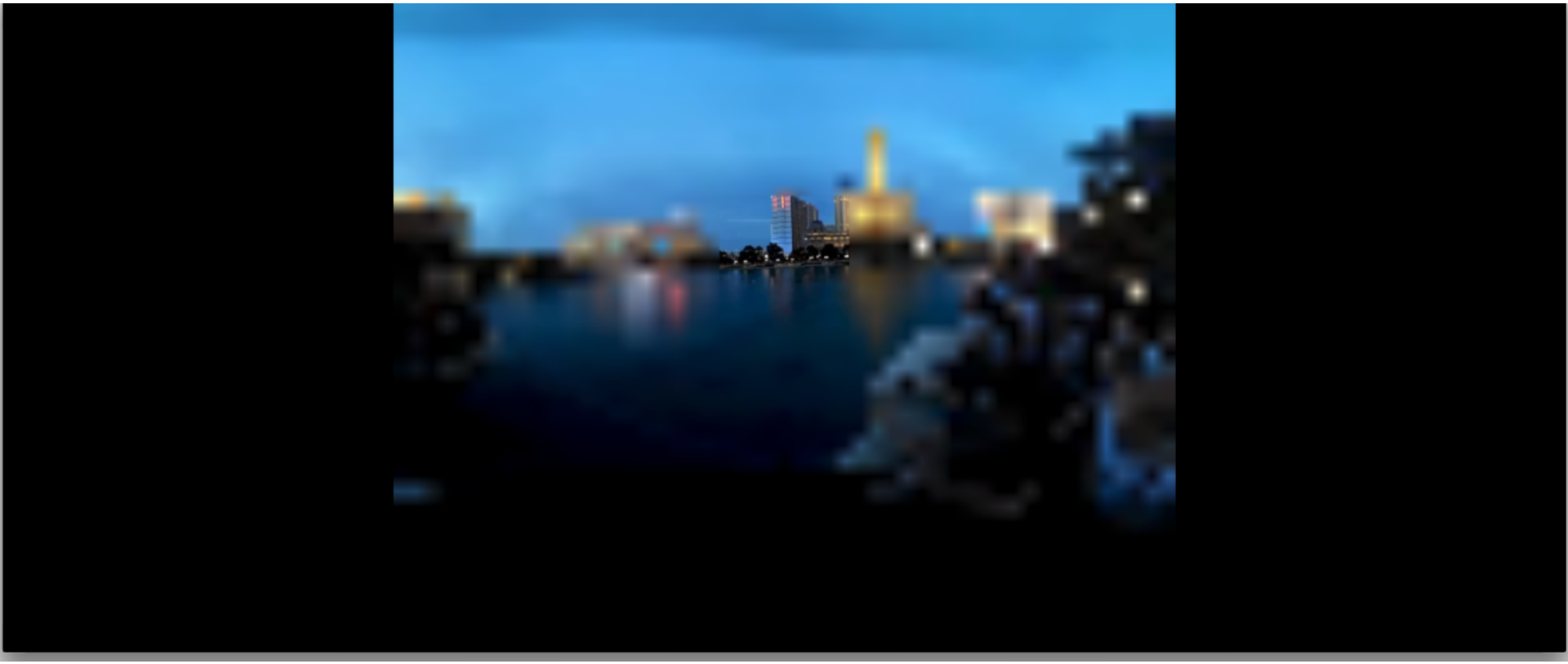
- Inputs
  - Decoded frame of a high resolution 360° 3D video
  - Orientation of the viewer's head
- Output
  - View optimized image frame, blurred around edges

Original image

360° 4K Panorama (one eye)

Crop out the area behind you

Blur unfocused outer regions



Human eye is lower resolution in outer region

*Blur increased 7x for demonstration

10% compression rate

- Crop frame to 50%, then shrink to 1/9 the size
- 50° x 50° uncompressed focus window
  - Only ~3% of original frame size
- **Result: ~10% size of original**
  - In terms of raw pixels

Conduit makes VR streaming practical

- 4K 3D Video: **18 Mbps** (H.264 encoded)
  - More to stream reliably!
- Average Internet speed (US): **11 Mbps**
- 4G: **8 Mbps**
- Conduit: **1.8 Mbps***
- We only use **~10%** of the bandwidth*

*Assuming video compression is linear

Bandwidth-latency tradeoff

- Decreased bandwidth, but...
- **Motion-to-update (MTU) latency**: time from when you move your head, until you see the updated view-optimized frame on the HMD.
- Internet latency now matters!
- View optimization adds **additional** latency
- To compensate, **predict** the user's head pose and **optimize** the entire pipeline.

Latency results

- Unoptimized motion to update latency: **150 ms**
- Optimized motion to update latency: **75 ms**
- Average frame rate: **50 FPS = 19 ms/frame**
  - Display: 12 ms
  - Load textures: 7 ms
  - Video reading/Optimization: 0 ms (latency hiding)
- Run on a Samsung Chronos 7
  - NVIDIA GT 630M
  - Intel Core i7 2.3 GHz with 4 cores

Optimizations

1. Buffered video loading in separate thread.
2. Render thread takes frames if video available, doesn't wait
3. Using Pixel Buffer Objects to stream asynchronously to OpenGL textures
4. View-optimization in a separate thread

Many minor optimizations, e.g. simplifying data structures, optimizing OpenGL API usage

Demo

Bellagio Fountains

Challenges with implementation

- Oculus runs unreliably
  - Sometimes display takes 10ms, sometimes 100ms
- Over 20 hours of build issues
  - Tried 4 different versions of Ubuntu, Mac, and also Windows across 2 computers
  - Could use the “simulator”, but would miss out on key qualitative data, like how noticeable blur and lag was.
- Documentation
  - Oculus SDK not fully documented

Things we didn’t have time to do

- OpenCV with CUDA and OpenGL support
- CUDA video decoder
- Client: resize image on graphics card, don’t go through CPU
- Profile with NSpire
- Multithreaded optimizer
- Resubmit command buffers using global constants
- Figure out performance leak in Oculus
- Test 4000x4000 video for fixed-function hardware
- OpenCV as native RGB, not BGR to avoid swizzle
- Copy only cropped portion of image
- Asynchronous timewarp
- Server-Client setup
- Much more...

Questions?

Thank you