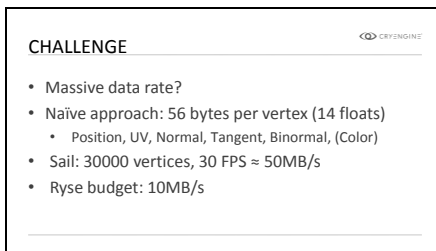


Animations like cloth, water simulations, fur  
Alembic: No engine specific markup. One click to import and run. Outsourcing much easier

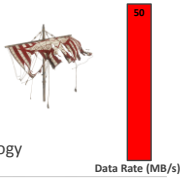


More than Alembic actually, need full tangent frames  
10 MB/s is for the whole scene, not only one cache

CRYENGINE

### COMPRESSION OVERVIEW

- Only transform for rigids
- Vertex animation
  - Fewer bits per vertex
  - Transform data
  - Compression
- Restriction: Only static topology



50  
Data Rate (MB/s)

Obviously we don't store per frame data for rigid / non-deforming meshes

Data rate is bar is always for the sail

Transform data to help compression

Data rate meter for specific sail asset with 30.000 vertices. Data rate meter will indicate progress


on data rate reduction during the methods

presented in the talk.

CRYENGINE

### TRANSFORMS

- Simplify the tree
- Transforms: Rotation + Translate + Scale



50  
Data Rate (MB/s)

Bake down static hierarchies

Bake down animated child of animated parent

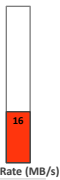
No support for shear

Transforms: 40 instead of 48 bytes. Compared to vertex animations we can neglect this. We didn't optimize it a lot.

CRYENGINE

### FEWER BITS (QUANTIZATION)

- Positions: 3x uint16
- Texture Coordinates: 2x int16
- QTangents 4x 10 bits (int16) [FREY11]
- (Colors: 4x 8 bits RGBA)
- Only lossy step



16  
Data Rate (MB/s)

Positions defined in bbox space. mm accuracy for 64m mesh. Quantizer will use less bits if possible, artist can specify the mm precision he needs.

Texture coordinates get mapped to [-1024, 1024] which leaves enough fractional digits

Tangent frames mean only orthonormal tangent frames. Doesn't matter in practice for us. We use 16 bit shorts for 10 bit values because compressor works on bytes

CRYENGINE

### COMPRESSION

- Block compress each frame
- Deflate (zlib) [DEUTSCH96] or LZ4 HC [COLLET13]

Data Rate (MB/s)

Deflate is slow but pretty good compression  
 LZ4 HC is usually 20% worse compression, but  
 10x faster decode. Almost like memcpy.  
 Still 10MB/s for one asset, we need to do better

CRYENGINE

### PREDICTION OVERVIEW

- Predict utilizing temporal and spatial similarities
- Store residuals (differences)

Data Rate (MB/s)

Of course need to do same prediction at runtime  
 than at compile time, otherwise don't get same  
 result  
 Residual symbols cluster around zero, because  
 prediction tends to be close. The more of the  
 same symbols, the better the compression.

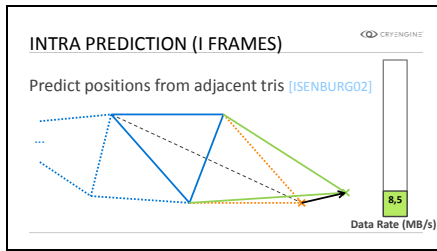
CRYENGINE

### PREDICTOR REQUIREMENTS

- Deterministic
- In-place prediction
- As fast as possible

Data Rate (MB/s)

Always needs to predict exactly the same way  
 (obvious)  
 No extra memory allocations on decode  
 Needs to decode in real time



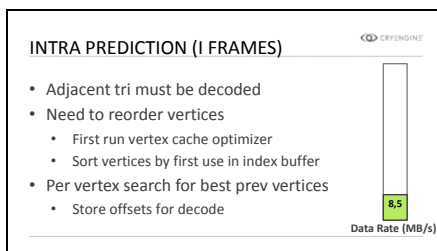
Parallelogram prediction. Extend adjacent triangles to parallelograms.

Blue: Last triangle(s) Orange: Parallelogram prediction Black arrow: Residual to store  
Can do this in place: for each vertex predict, read, add, write

Also used for UVs

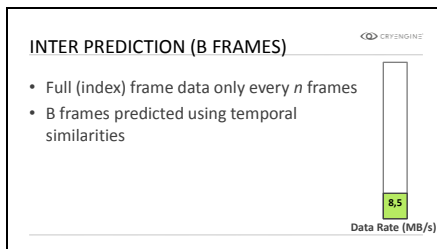
QTangents just use average of last two vertices, because parallelogram rule makes no sense for them

Savings depend on asset

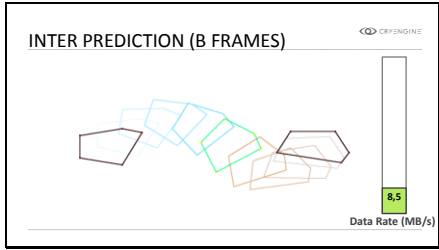


Optimizer tends to order indices so mesh gets rendered in strips

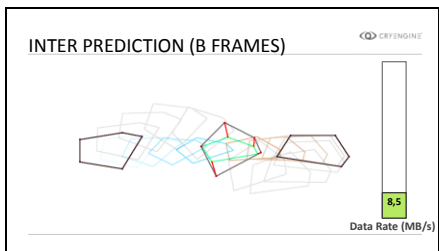
Offset only needs to be stored in file header, because we only support non-changing topology



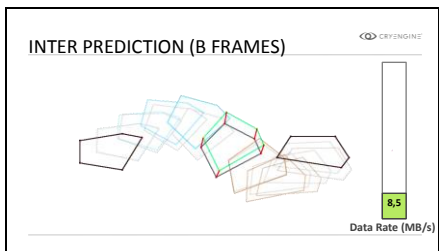
For us optimal index frame distance was about 10



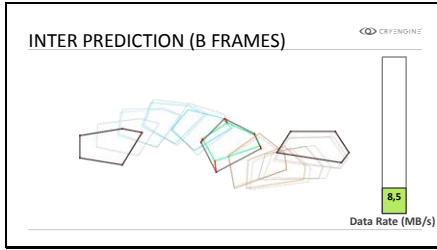
Original motion



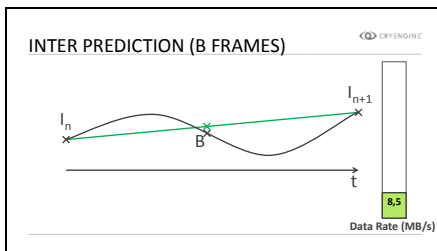
Interpolated prediction



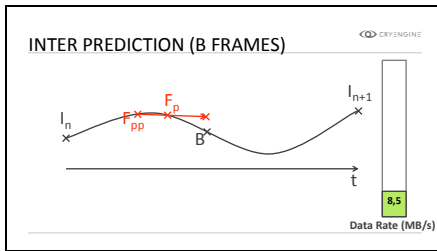
Motion prediction



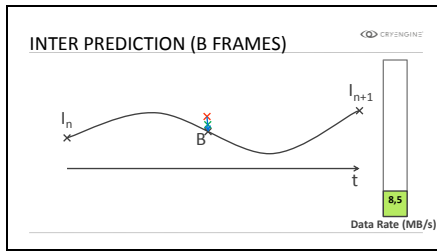
Combination of motion and interpolation prediction



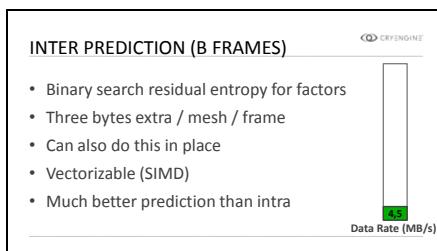
Select interpolation factor for interpolation



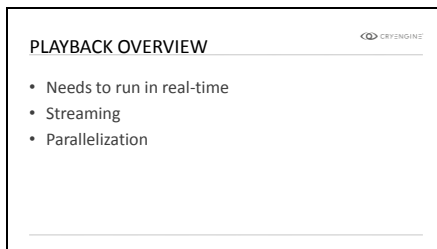
Select acceleration factor for motion



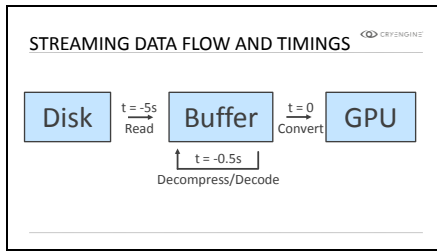
Select extrapolation factor for combination  
 The three factors used are the same for all vertices in mesh, so individual predictions will usually be worse than here.



Factors get quantized to three bytes per frame for all vertices  
 Can use SIMD for this to predict 4 elements in parallel (some INT16 operations even on 8 elements)  
 Just unpack 8xINT32 -> 2x4xUINT32, mul, shift, truncate & pack again on Jaguar per interpolate/extrapolate



Loading time would also be a problem  
 Next gen CPU cores still not terribly fast

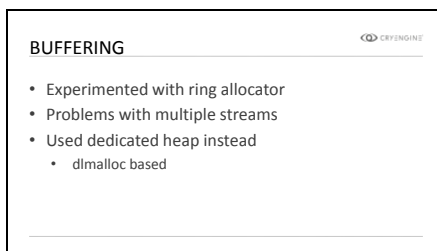


Disk reads and decompress/decode are asynchronous and non-blocking  
 Read combining to avoid disk seeks (>1 MB chunks)

Upload to GPU is asynchronous but render thread will wait for data

Data in buffer stays in compact disk format until decompress starts

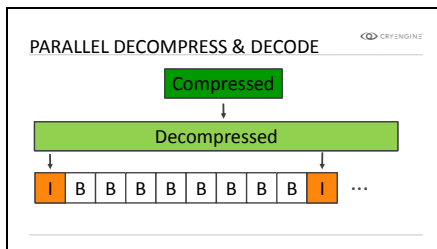
Timings are configurable, values were chosen by experimentation



Ring buffer allocator doesn't work for multiple streams, because of different data rates. Would need to defragment holes. Really tried to make this work, but wasn't worth it in the end.

Fragmentation with normal allocator wasn't a big problem

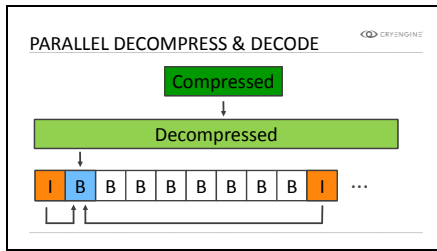
128 MB for both compressed and uncompressed data



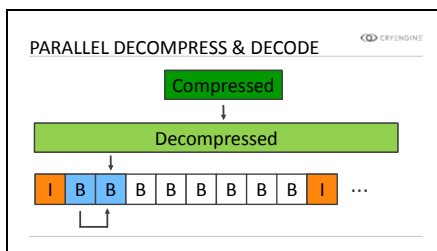
Jobs for decompression

Index frame jobs can start as soon as decompressed data is ready





First B frame has I frames and own residuals as input (First B frame does not do acceleration prediction)

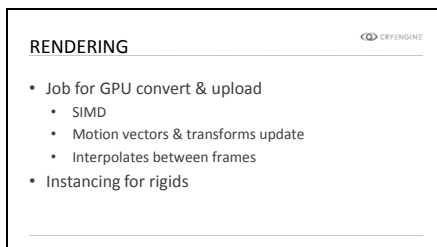


Other B frames have only last B frame and residuals as dependencies, because by induction both I frames and the last two B frames are decoded

All other B-frames depend on previous one as well

We do the synchronization for all jobs with lockless atomic counting:

- I frames get initialized to 1, first B frame after I frame to 3, all other B frames to 2
- After dependency is finished will decrease counter of dependent task and launch it when counter reaches 0



The conversion job gets launched for each job the geom cache actually gets rendered

Conversion job could possibly be avoided if GPU would directly read quantized format

The vertex shader could possibly directly support the quantized format

## FUTURE DEVELOPMENT



- Support for changing topology
- Improve compression
  - Better predictors
  - Better block compression
  - Automatic skinning
- Support for physics
  - Tricky with vertex animation

All of the compression research tends to lead to require more and more computational power for little gains

Automatic skinning would be a way to do more lossy compression

## SPECIAL THANKS



- Sascha Herfort, Nicolas Schulz, Bogdan Coroi, Theodor Mader, Carsten Wenzel, Chris Raine, Chris Bolte & Ivo Zoltan Frey
- The entire Ryse team and Crytek

## REFERENCES



- [DEUTSCH96] DEFLATE compressed data format specification version 1.3.
- [COLLET13] Collet, Yann. "LZ4: Extremely fast compression algorithm." <https://code.google.com/p/lz4/>
- [FREY11] Frey, Ivo Zoltan, and Ivo Herzog. "Spherical skinning with dual quaternions and QTangents." *SIGGRAPH Talks*. 2011.
- [ISENBURG02] Isenburg, Martin, and Pierre Alliez. "Compressing polygon mesh geometry with parallelogram prediction." *Visualization, 2002*