Simple and Powerful Animation Compression

Nicholas Fréchette
Programming Consultant
for Eidos Montreal

Contributors

- Frédéric Zimmer, co-designer
- Luke Mamacos, consultant

- Thank you Eidos Montreal!

Presentation outline

- A bit of context

Presentation outline

- A bit of context
- The classic solutions

Presentation outline

- A bit of context
- The classic solutions

- Our special blend!

A bit of context

- The game engine used by Rise of the Tomb Raider (2015)

- Same compression algorithms since ~1996!
  - Used in dozens of AAA titles
  - Based on linear key reduction

A bit of context

\+ Good size

\- Slow decompression

\- Sub-par accuracy

The problem

- Cinematic clips
  - Need high accuracy
  - Size matters

The problem

- Cinematic clips
  - Need high accuracy
  - Size matters
- Ever higher need for accuracy
  - 40% clips used weak compression
  - Time is the enemy

The problem

- Legacy code not ideal
  - Very old, aged poorly
  - Not streaming friendly
  - Nobody wants to get near it

Design goals

- Solve cinematics first
  - Through streaming (if we need to)
- Keep it simple
  - Time budget: 20 days

Design goals

- Fast decompression
- High accuracy

Design goals

- Best effort
  - Small size
  - Nothing to tweak
  - Supersede everything

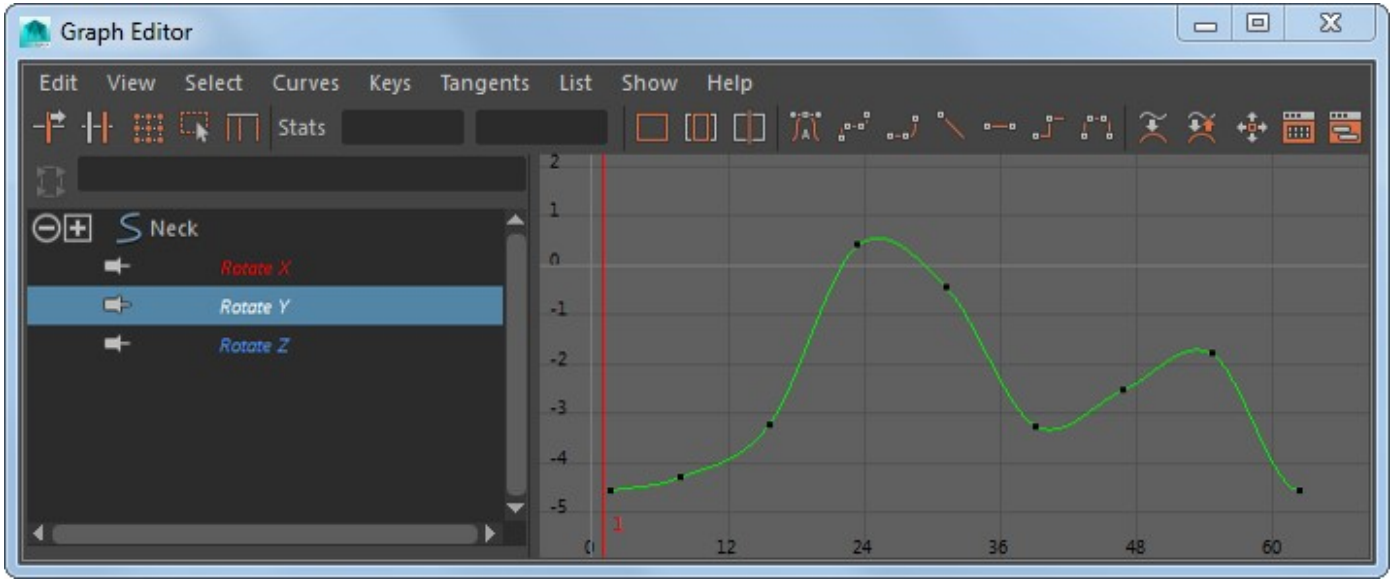
Most common algorithm families

- Signal processing
- Curve fitting
- Linear key reduction
- Simple key quantization

Signal processing

- E.g. wavelets
- Too complex
- See blog!

Curve fitting



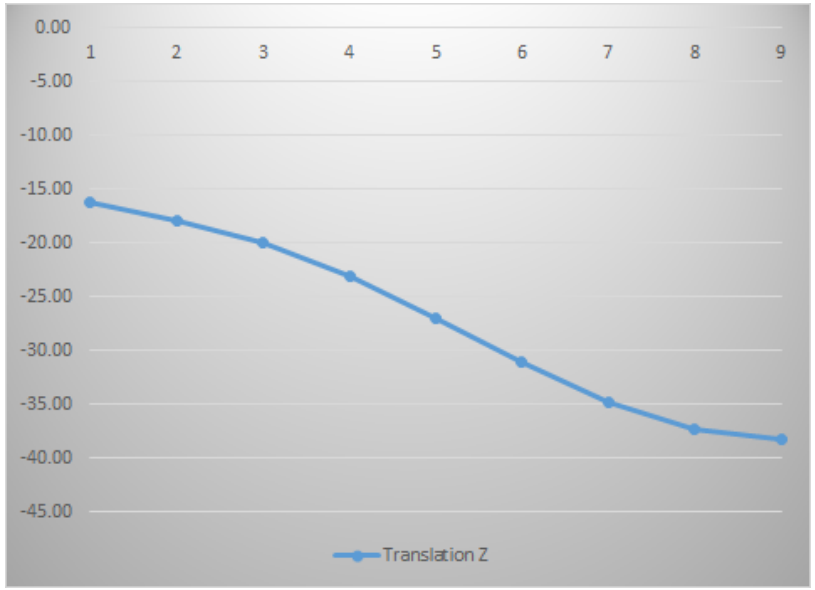
Curve fitting

- Pros
  - Sensible choice
  - Very compact

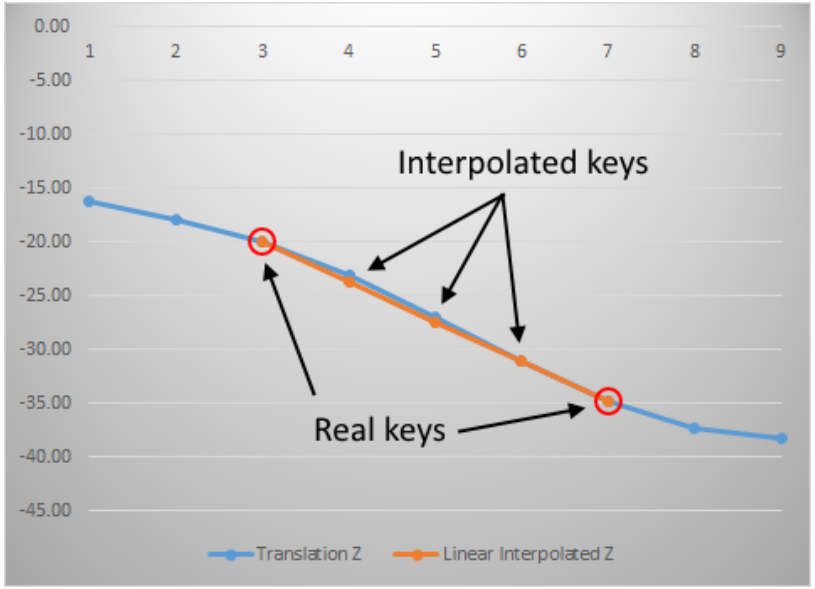
Curve fitting

- Cons
  - Not accessible
  - Slower decompression
  - Medium complexity
  - Not great for mocap

Linear key reduction



Linear key reduction



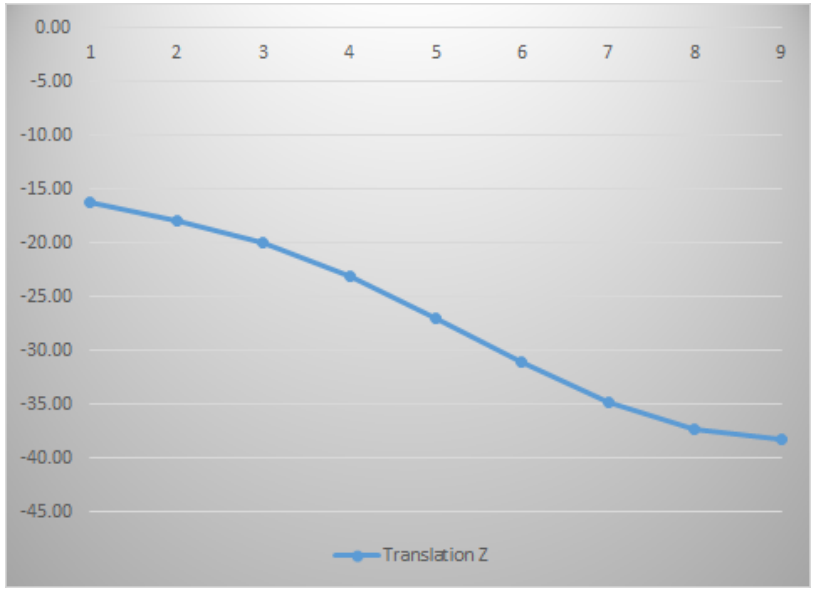
Linear key reduction

- Pros
  - Reasonably simple
  - Reasonably compact
  - Similar to legacy impl.

Linear key reduction

- Cons
  - Similar to legacy impl.
  - Slow decompression
  - Not great for mocap

Simple quantization



Simple quantization

- Pros
  - Dead simple
  - Very fast
  - Solid foundation

Simple quantization

- Cons
  - Not very compact

- **Good enough!**

Our solution

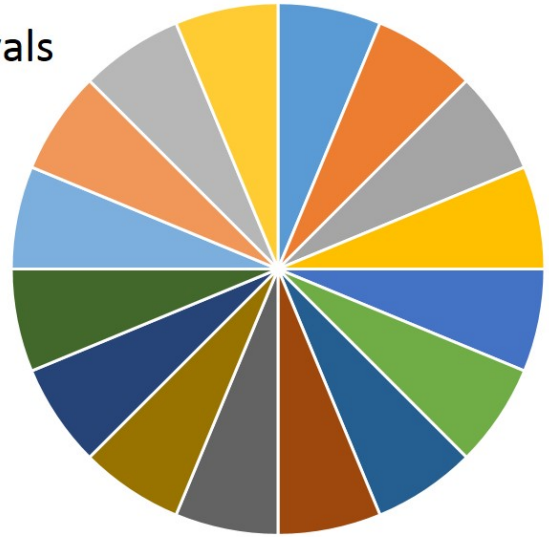
- Range reduction
- Uniform segmenting
- Constant tracks
- Quantization

Range reduction

- Tracks in terms of ranges
- E.g. my elbow rotates by 120° in a clip
  - Theoretical range: 360°
  - Effective range: 120°

Range reduction

4 bits = 16 intervals

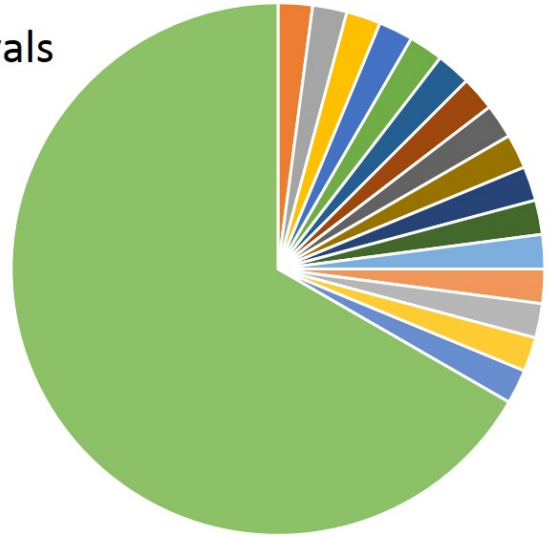


Precision: 22.5°

Range: 360°

Range reduction

4 bits = 16 intervals



Precision: 7.5°

Range: 120°

Range reduction

- Some overhead
  - Range Minimum
  - Range Extent
- Normalizes our values
- Range expansion is trivial
  - (normalized value * extent) + min

Range reduction

- 6 scalar values = full track range
  - 3 components * (min, extent)
- Clip metadata
  - 6x floats = 24 bytes
  - **per animated track**

Range reduction

- More accuracy => can be more lossy
- Bad for short clips

- **Worth it!**

Uniform Partitioning

- Split clips in blocks of 16 key frames

Uniform Partitioning

- Split clips in blocks of 16 key frames
  - Fast and easy seeking

Uniform Partitioning

- Split clips in blocks of 16 key frames
  - Fast and easy seeking
  - Easy streaming

Uniform Partitioning

- Split clips in blocks of 16 key frames
  - Fast and easy seeking
  - Easy streaming
  - Range reduction per block

Uniform Partitioning

- Block metadata
  - 6x 8 bit = 48 bits = 6 bytes
  - **per animated track**

  - ***Lossy normalization!!***

Constant Tracks

- For our main characters (~3500 clips)

Constant Tracks

- For our main characters (~3500 clips)
  - Bones: 65% constant, 45% bind pose

Constant Tracks

- For our main characters (~3500 clips)
  - Bones: 65% constant, 45% bind pose
  - Tracks: 87% constant, 79% bind pose

Constant Tracks

- Per clip, 1 bit **per track**:
  - Is it bind pose?
  - Yes? Drop it!

Constant Tracks

- Per clip, 1 bit **per track**:
  - Is it bind pose?
  - Yes? Drop it!
- Per clip, 1 bit **per track**:
  - Is it constant?
  - Yes? Keep 1 key! (3x floats)

Quantization

- Hard coded, 16 bits

Quantization

- Hard coded, 16 bits
- Best common rate per clip

Quantization

- Hard coded, 16 bits
- Best common rate per clip
- Best rot/trans/scale rate per clip

Quantization

- Hard coded, 16 bits
- Best common rate per clip
- Best rot/trans/scale rate per clip
- Best rot/trans/scale rate per block

Quantization

- Hard coded, 16 bits
- Best common rate per clip
- Best rot/trans/scale rate per clip
- Best rot/trans/scale rate per block
- ***Best individual track rate per block***

Variable bit rate

- Ideal for:
  - Hierarchical data
  - Exotic tracks
  - Temporal coherence
  - Everything!

Quantization details

- 16 possible bit rates

- Bit rates: 0, 3, 4, .., 16, 23

Quantization details

- 23 was a bad & naïve choice
  - Same as float mantissa minus sign bit

Quantization details

- 23 was a bad & naïve choice
  - Same as float mantissa minus sign bit
  - De-quantization requires our integer to be representable as a floating point number

Quantization details

- 23 was a bad & naïve choice
  - Same as float mantissa minus sign bit
  - De-quantization requires our integer to be representable as a floating point number
  - ***32 bit float = 6 significant digits*** ☹

Quantization details

- 23 was a bad & naïve choice
  - Same as float mantissa minus sign bit
  - De-quantization requires our integer to be representable as a floating point number
  - ***32 bit float = 6 significant digits*** ☹
  - 19 might be a better choice, measure!

Quantization details

- 0 => track is constant in block
  - We don’t need range information

Quantization details

- 0 => track is constant in block
  - We don’t need range information
  - Store our constant key instead!

Compression

- Almost everything is fairly trivial
- Only complex step is bit rate selection

Compression

- Almost everything is fairly trivial
- Only complex step is bit rate selection
  - Measuring accuracy
  - Need a smart heuristic

Measuring accuracy

- **Important!**
  - Our algorithm iterates with it
  - We compare our results to others with it

Measuring accuracy

- **Important!**
  - Our algorithm iterates with it
  - We compare our results to others with it
- Often overlooked and poorly implemented!

Measuring accuracy

- Three important criteria:
  - Account for hierarchy
  - Account for aggregate transform
  - Account for visual mesh

Measuring accuracy

- Hierarchy is important!
  - Error accumulates down hierarchy
  - Don’t use local space metrics!
  - Use object space

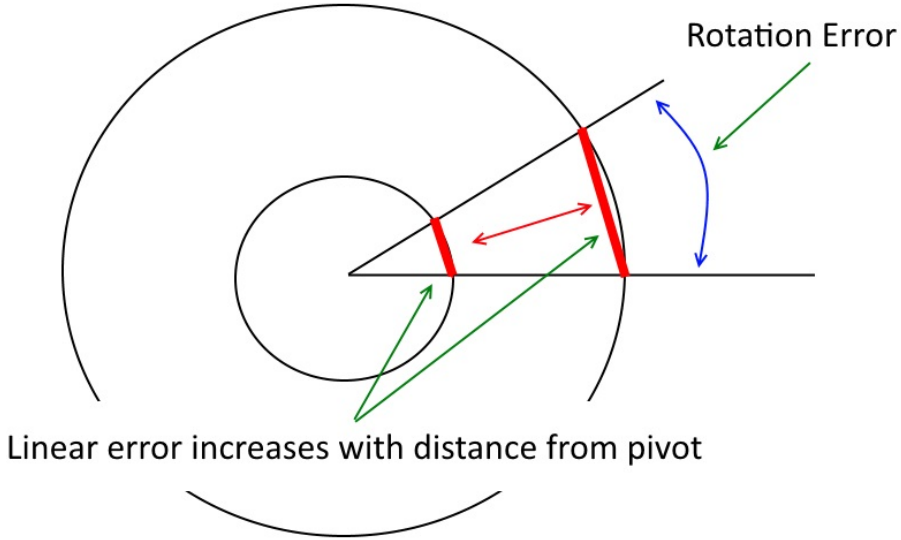
Measuring accuracy

- Aggregate error is important!
  - Don’t measure error with leaf bone position
  - It ignores rotation/scale contribution!

Measuring accuracy

- Skeleton error != visual mesh error
  - Skeleton is never visible, visual mesh is
  - With rotation & scale, error increases with distance from bone

Measuring accuracy



Measuring accuracy

- Vertex displacement on the visual mesh is the true measure of accuracy
  - Skinning == metric function
  - Satisfies all 3 criteria

Measuring accuracy

- But…
  - It is way too slow
  - Mesh information might not be available
  - Some bones have no skinned vertices

Measuring accuracy

- Use virtual vertices instead!
  - Approximates skinning
  - Satisfies all 3 criteria
  - Intuitive tweaking: distance from bone
  - Output is object space displacement error

Measuring accuracy

- We use:
  - 3 cm for normal bones
  - 1 m for high accuracy bones

Bit Rate Selection

- Huge search space!
  - Need smart heuristic

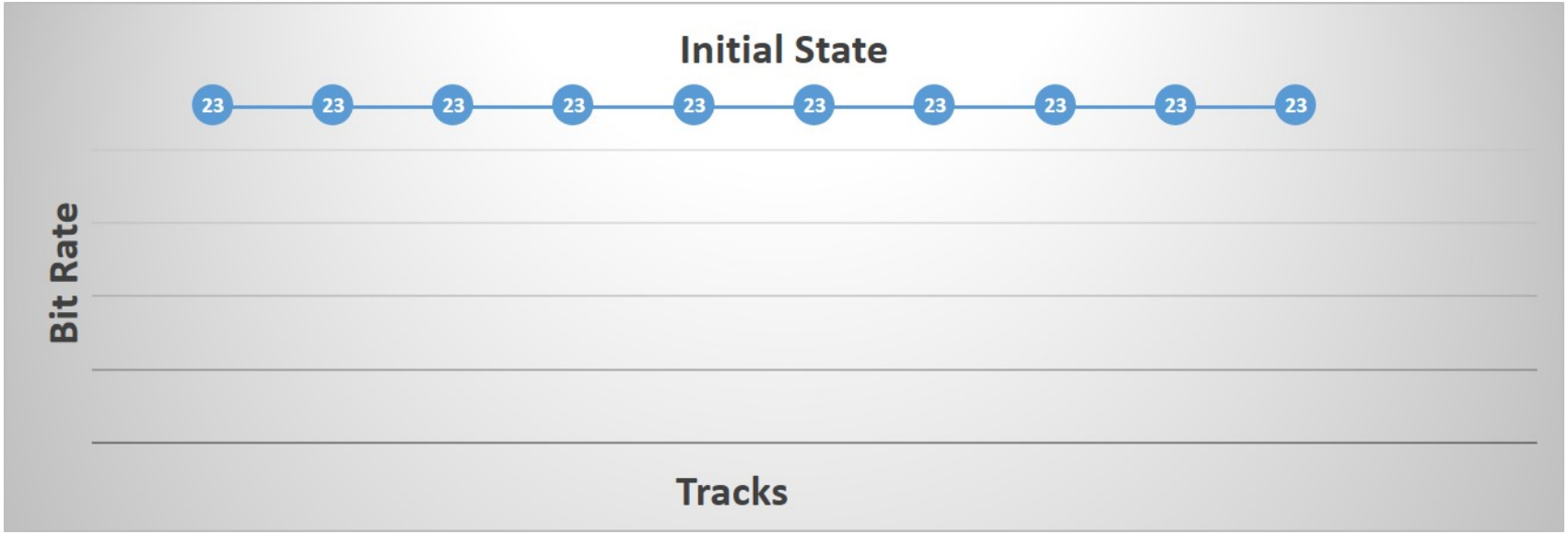
Bit Rate Selection

- Huge search space!
  - Need smart heuristic
- First pass finds an approximate solution

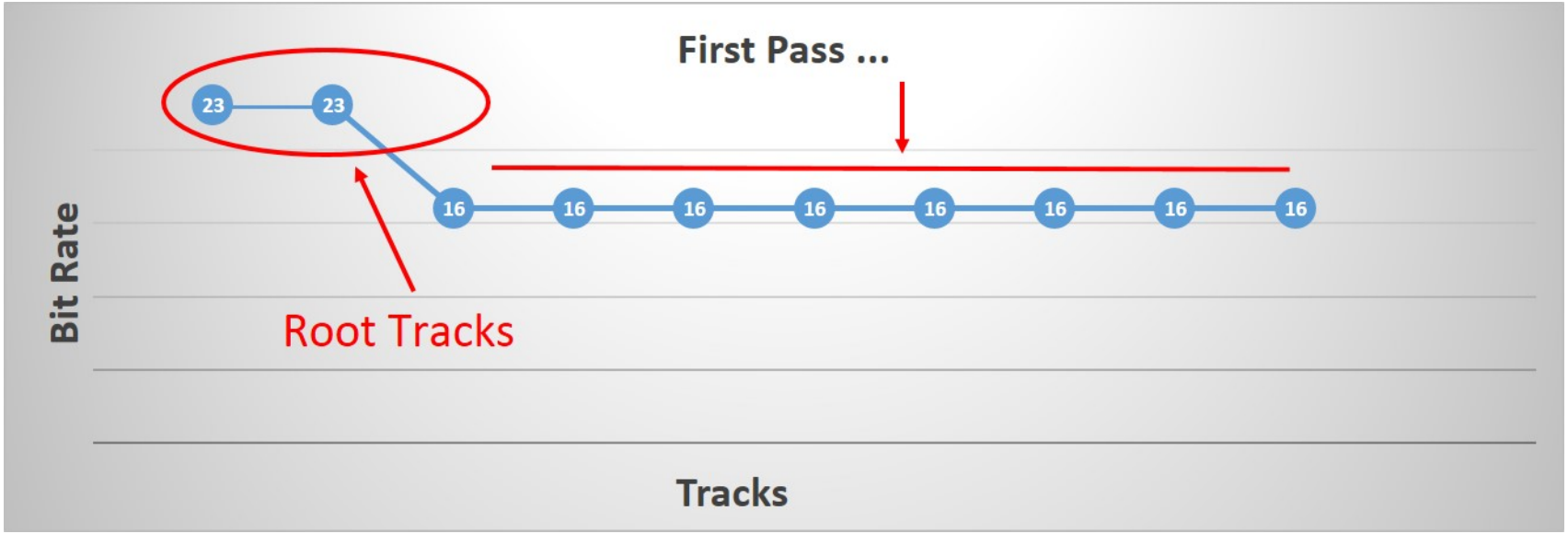
Bit Rate Selection

- Huge search space!
  - Need smart heuristic
- First pass finds an approximate solution
- Second pass refines to local minimum

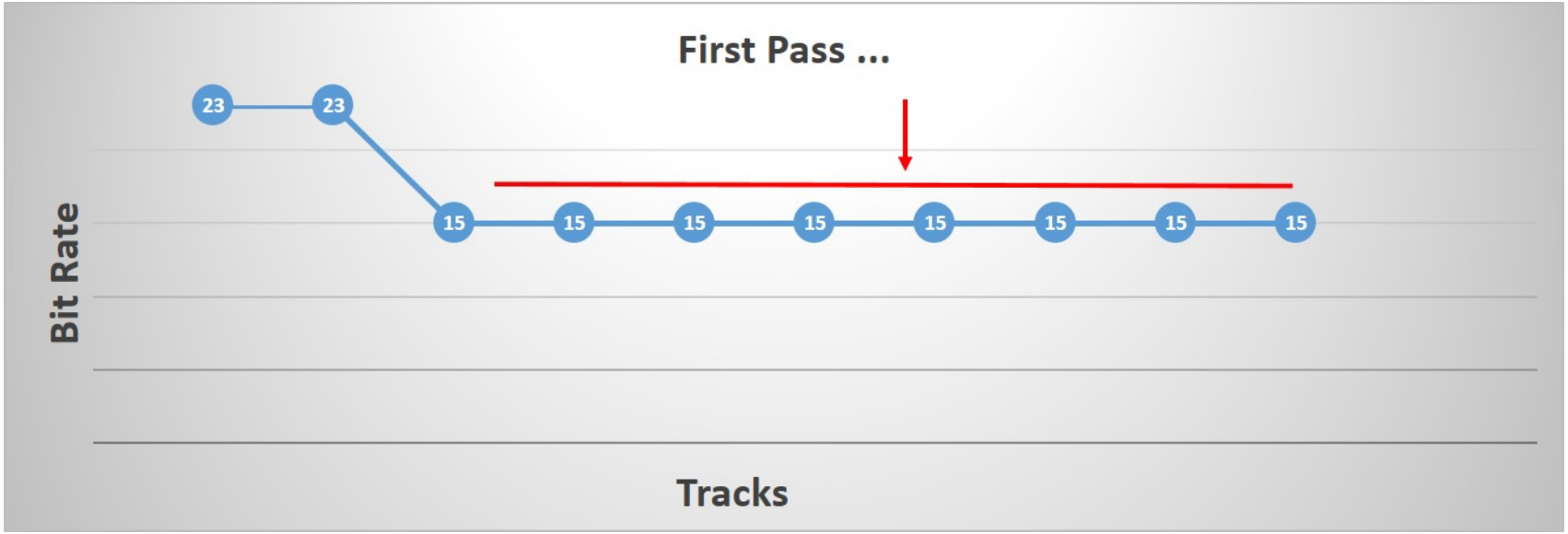
Bit Rate Selection



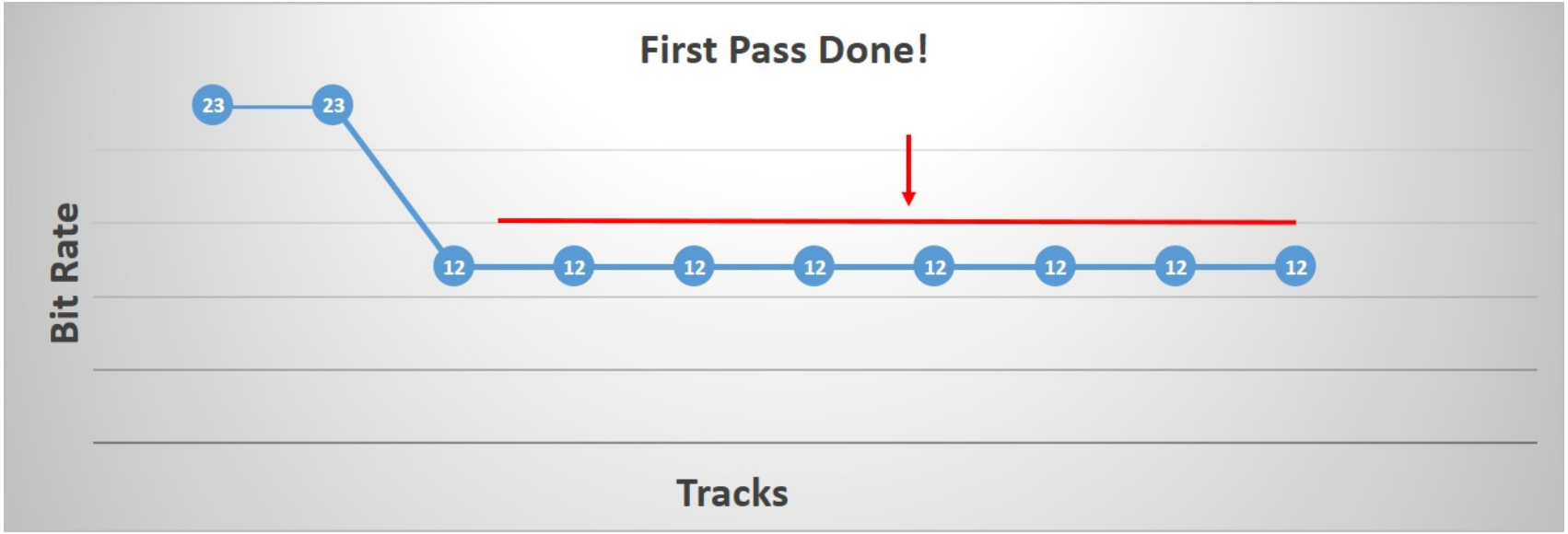
Bit Rate Selection



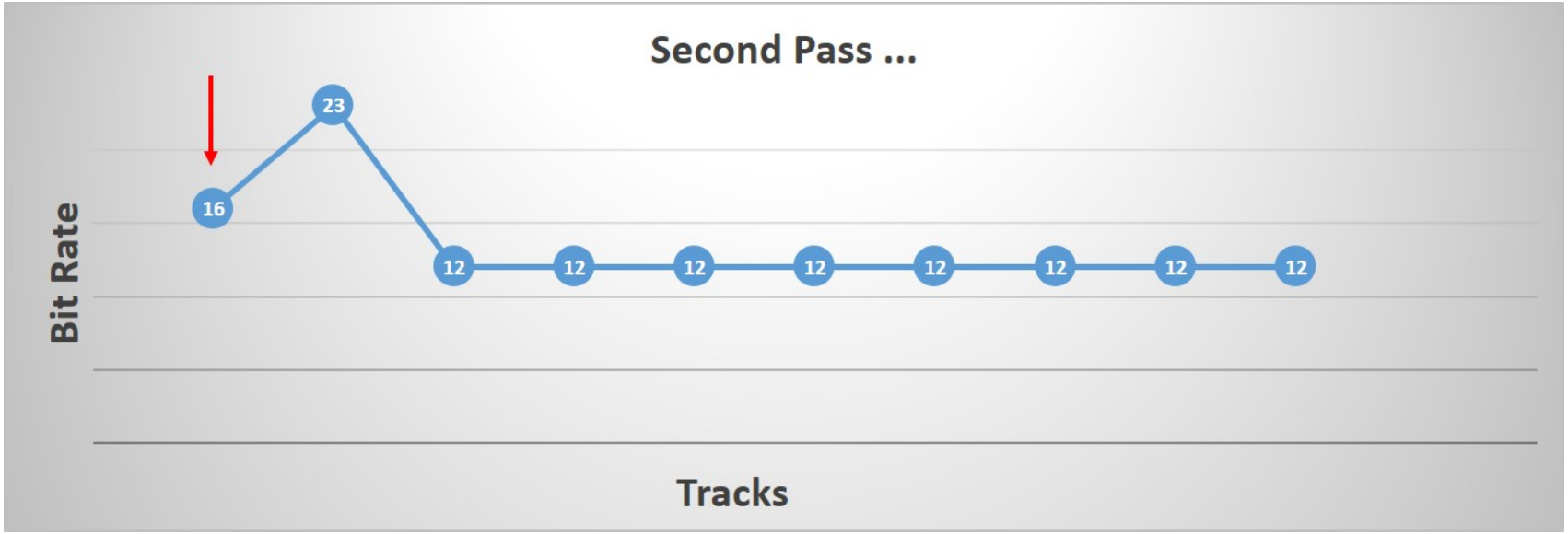
Bit Rate Selection



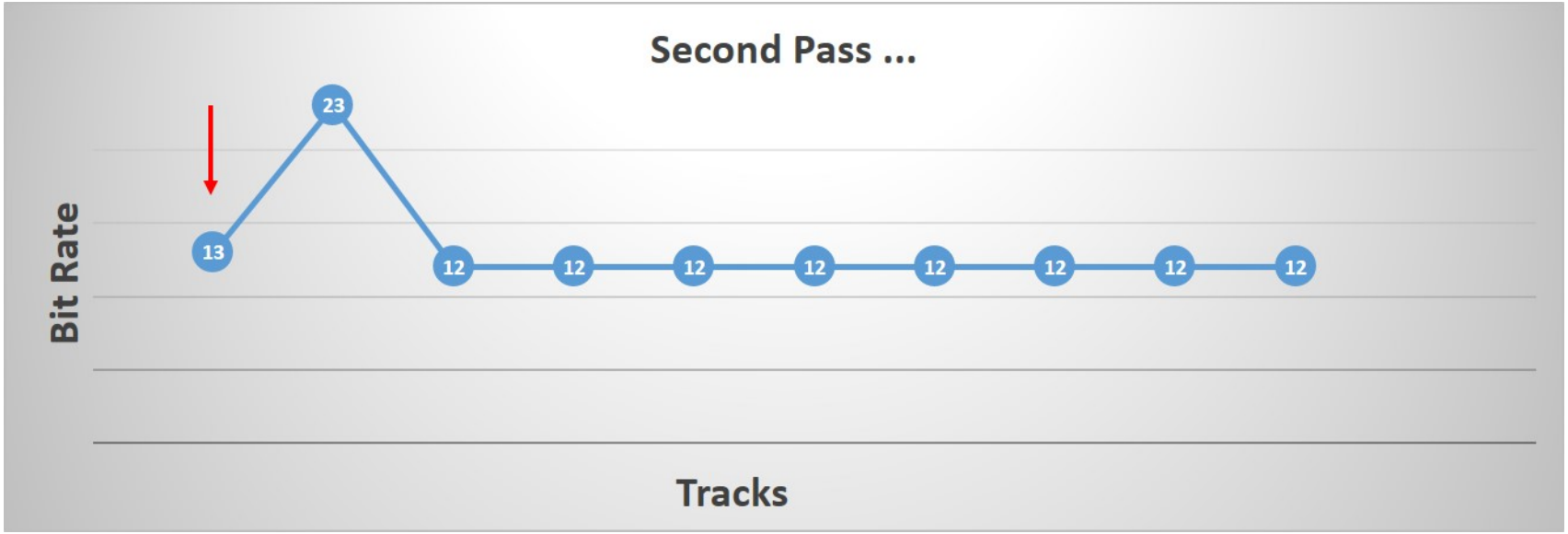
Bit Rate Selection



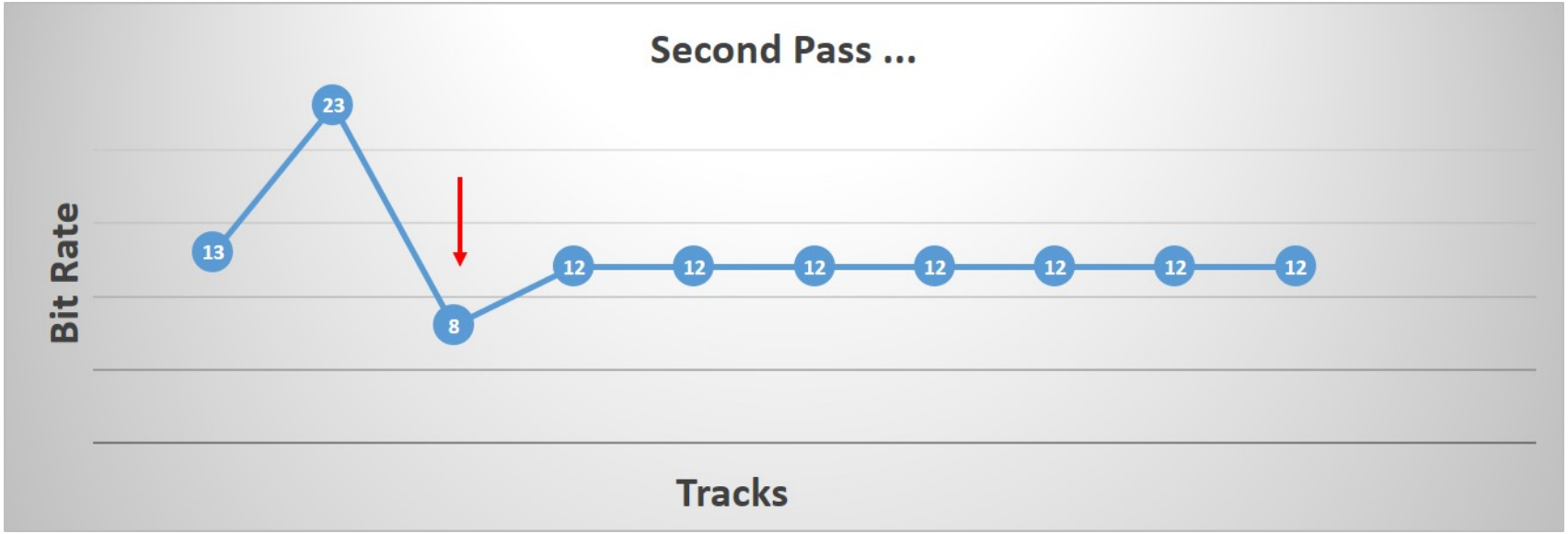
Bit Rate Selection



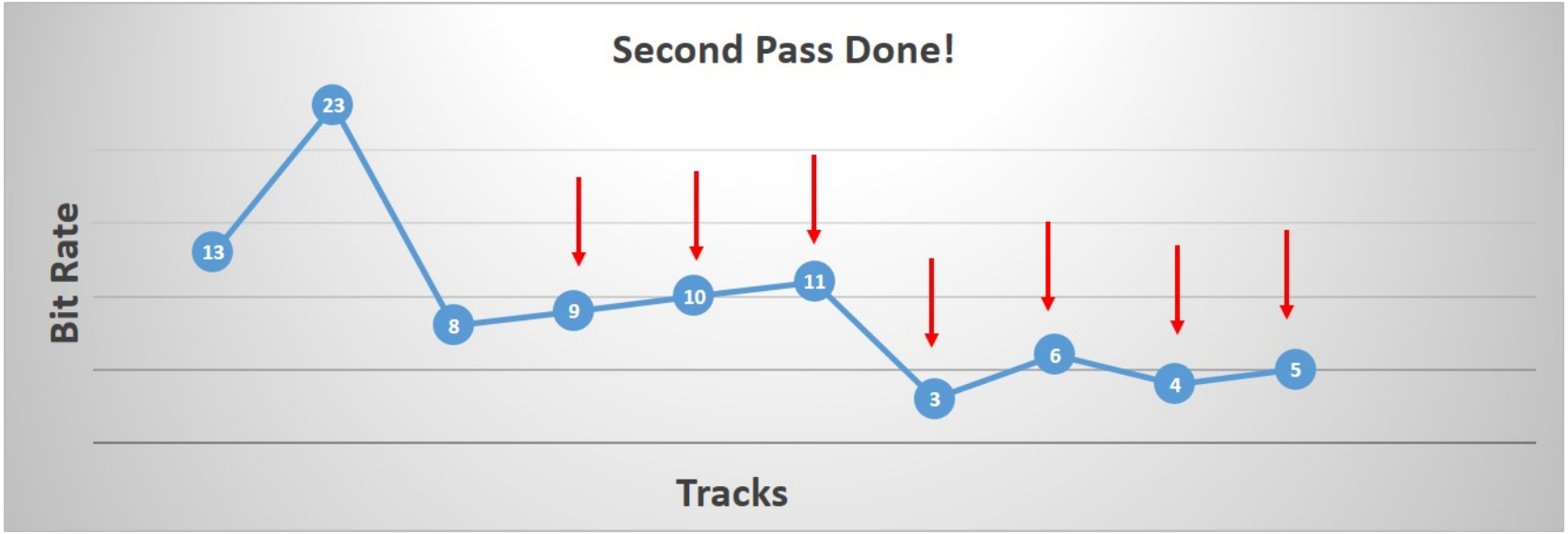
Bit Rate Selection



Bit Rate Selection



Bit Rate Selection



Bit Rate Selection

- We minimize the bit rate
- We maximize the error

- Threshold is important!
  - 1 mm is too high

Bit Rate Selection

- Hardcoded threshold: ***0.1 mm***
  - **Sub-millimeter accuracy!**

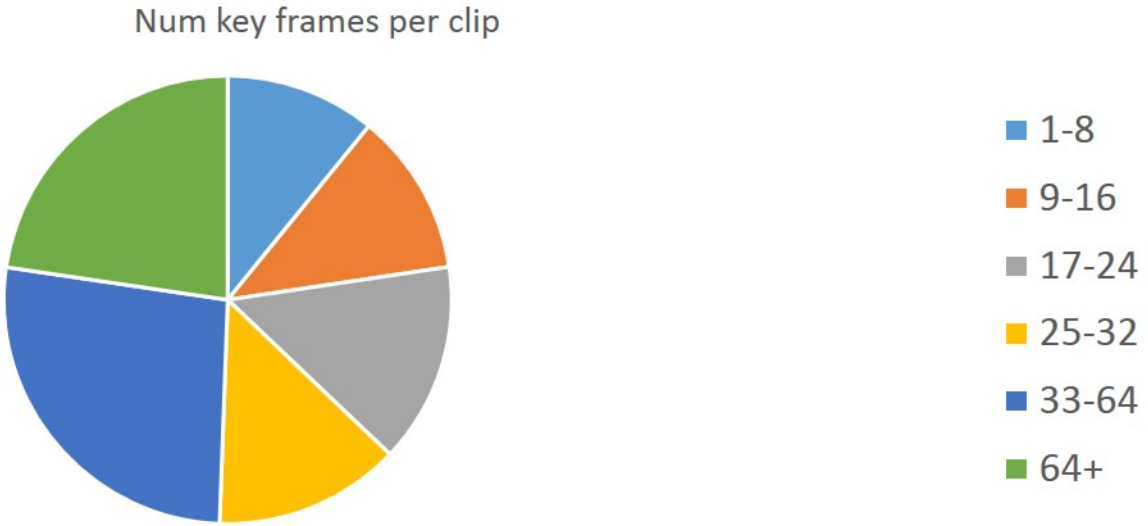
The results

- Aggregated
- Concrete examples

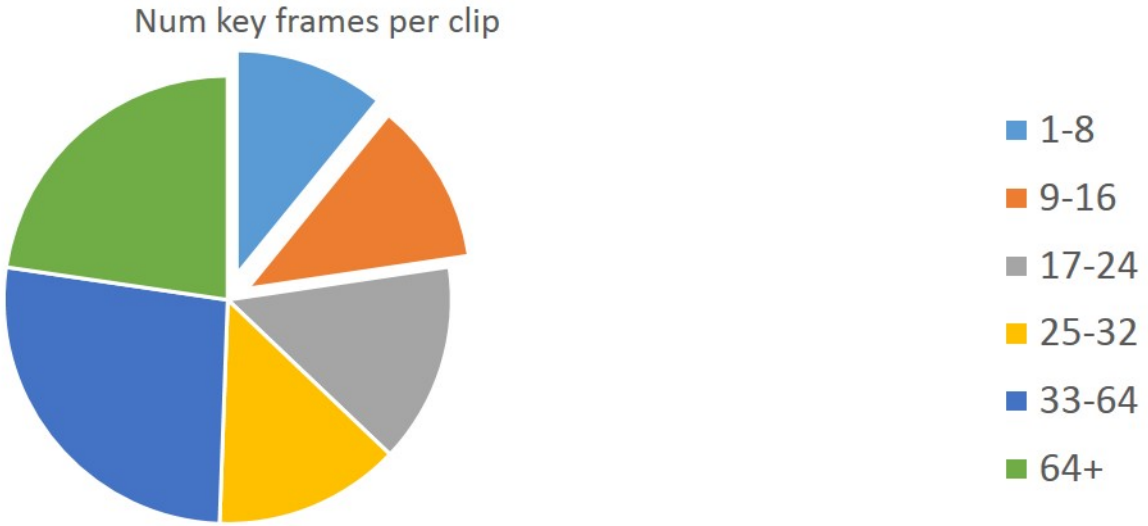
Aggregate Results

- 3900 clips (various characters)
- Compression time:
  - 3.2 hours, single threaded
  - 10 minutes, multi threaded
- Sum of clip lengths:
  - 5.4 hours @ 30 FPS

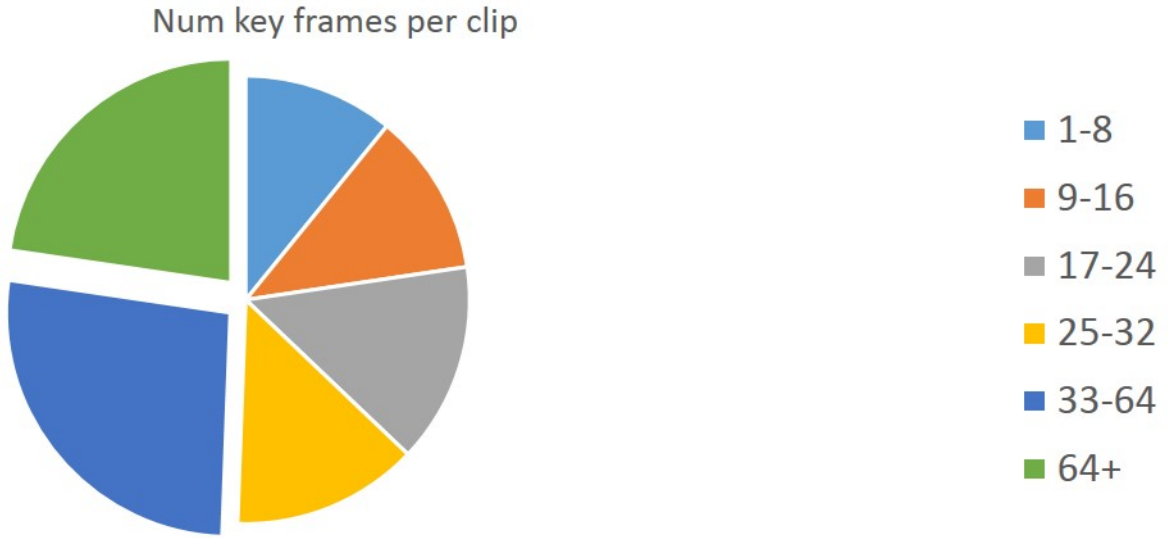
Aggregate Results



Aggregate Results



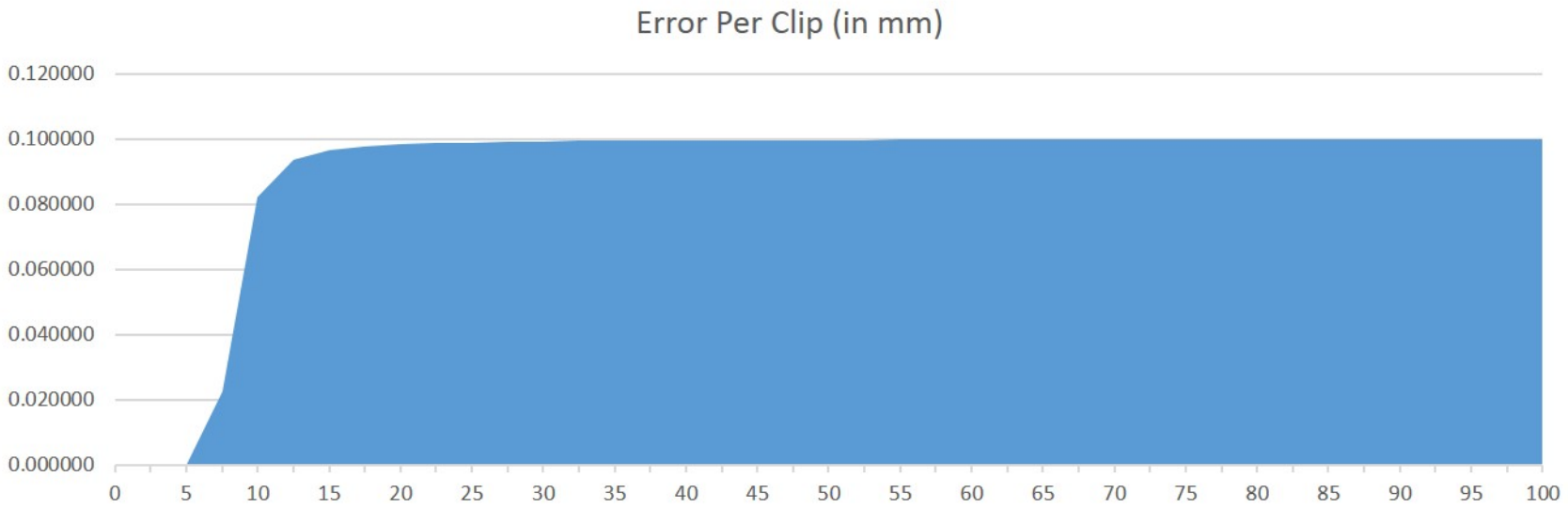
Aggregate Results



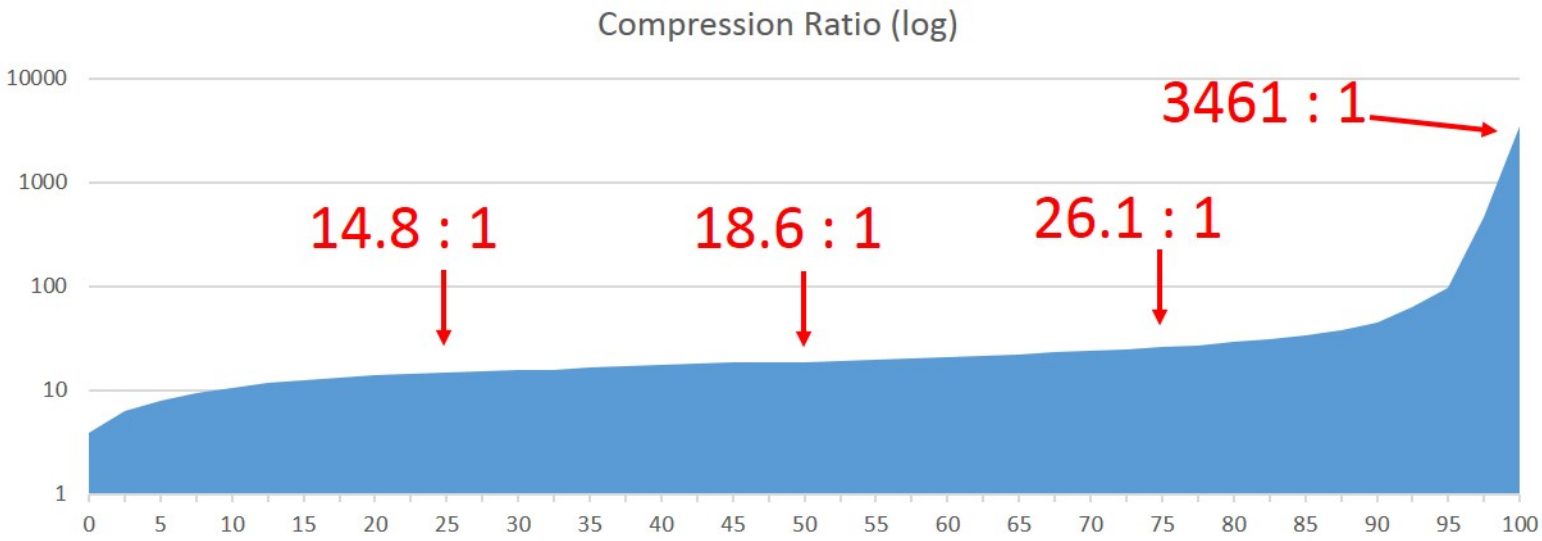
Aggregate Results

- Total size on disk: 168.4 MB
  - Legacy size: 300.0 MB
- Average number of animated tracks:
  - 56.0 per clip

Aggregate Results



Aggregate Results



Raw Clip Size

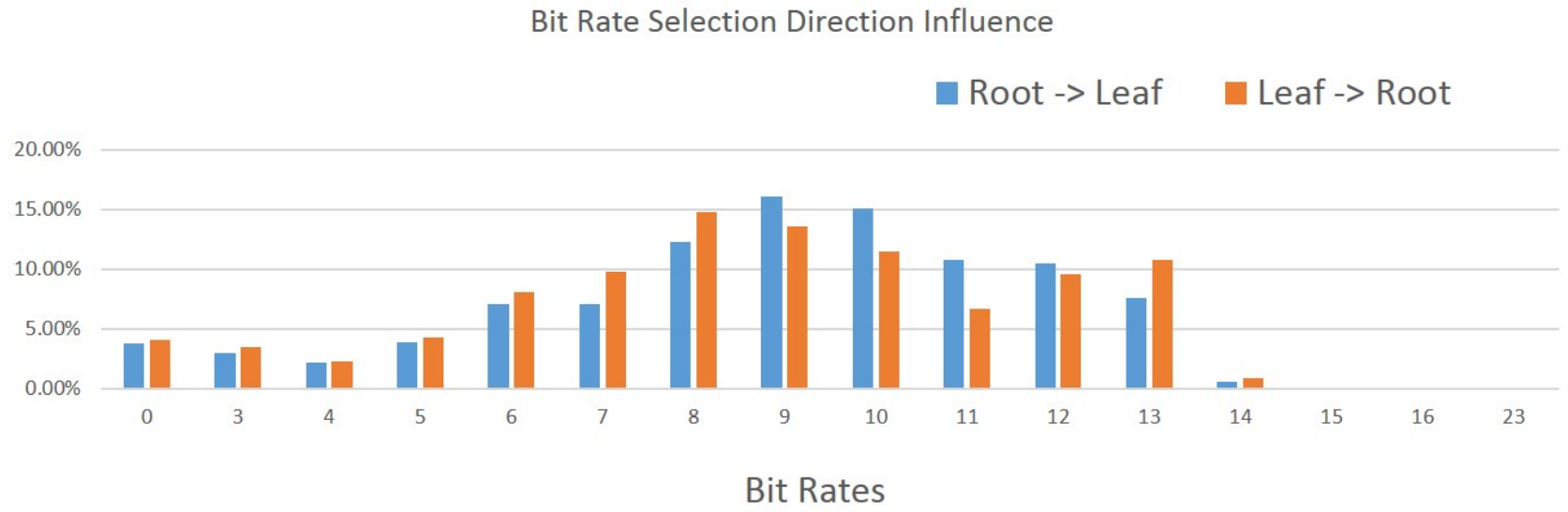
- Compression ratio is meaningless
- Unless raw size is consistent!

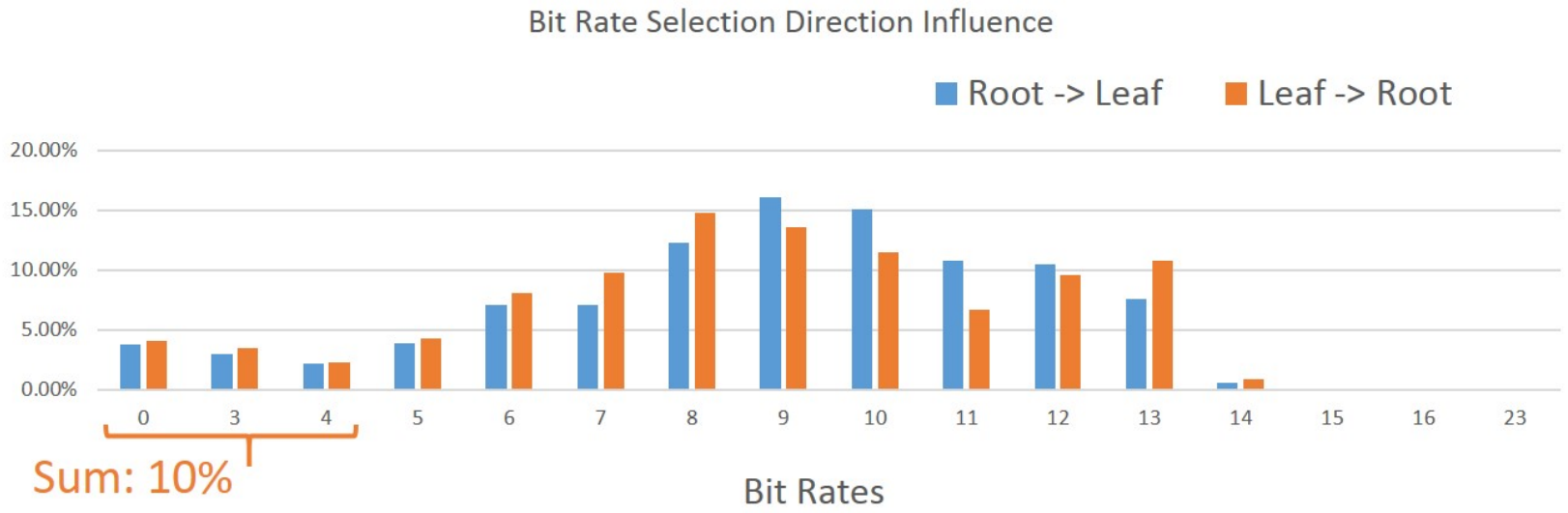
Raw Clip Size

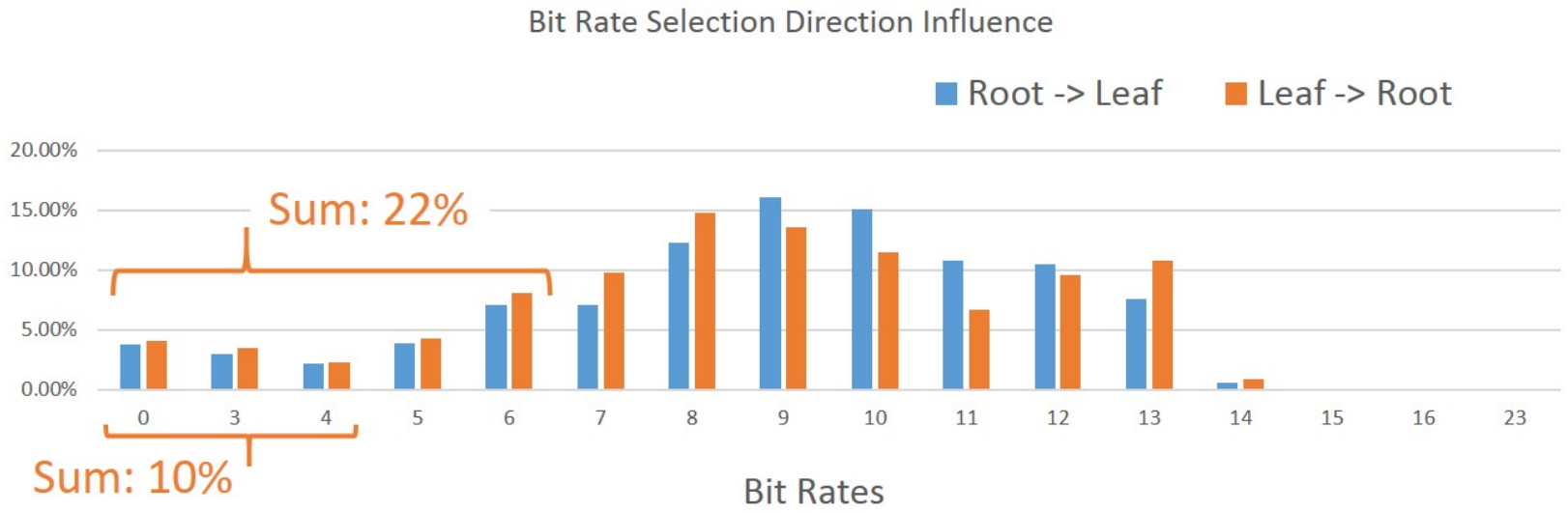
- Raw size =
  # key frames * # bones * 36 bytes

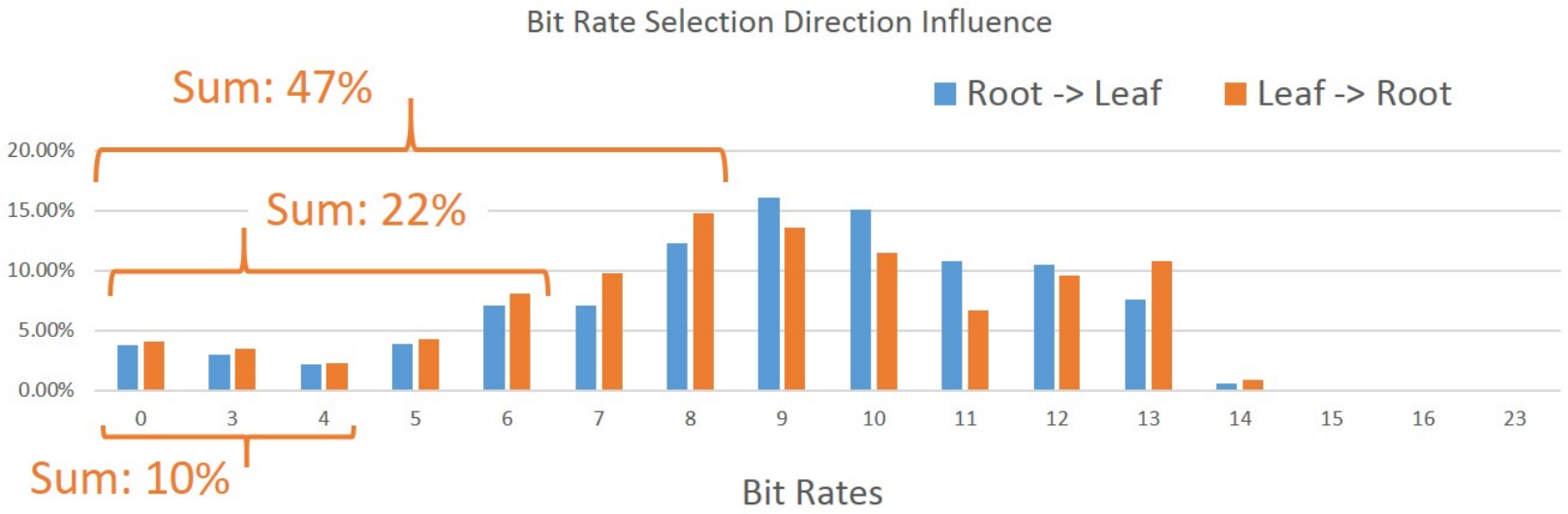
Raw Clip Size

- 1 track key = 3 floats = 12 bytes
- 1 bone key = 3 track keys = 36 bytes
- 1 key frame with 140 bones = 140 bone keys = 4.9 KB
- 30 key frames = 147.7 KB


Bit Rate Selection Direction Influence
Root -> Leaf
Leaf -> Root
20.00%
15.00%
10.00%
5.00%
0.00%
0
3
4
5
6
7
8
9
10
11
12
13
14
15
16
23
Bit Rates


Bit Rate Selection Direction Influence
Root -> Leaf
Leaf -> Root
20.00%
15.00%
10.00%
5.00%
0.00%
0
3
4
5
6
7
8
9
10
11
12
13
14
15
16
23
Sum: 10%
Bit Rates


Bit Rate Selection Direction Influence
Root -> Leaf
Leaf -> Root
20.00%
15.00%
10.00%
5.00%
0.00%
Sum: 22%
Sum: 10%
0
3
4
5
6
7
8
9
10
11
12
13
14
15
16
23
Bit Rates


Bit Rate Selection Direction Influence
Root -> Leaf
Leaf -> Root
Sum: 47%
Sum: 22%
Sum: 10%
20.00%
15.00%
10.00%
5.00%
0.00%
0
3
4
5
6
7
8
9
10
11
12
13
14
15
16
23
Bit Rates

Some concrete examples

	Scramble	Idle	Walk	Cinematic
Num. key frames	24	80	38	2291
Num. animated tracks	56 (11%)	77 (15%)	82 (16%)	87 (16%)
Raw size	147 KB	489 KB	232 KB	14498 KB
Compressed size	7 KB	15 KB	12 KB	521 KB
Compression ratio	20 : 1	32 : 1	19 : 1	28 : 1
Avg key frame size	314 B	194 B	325 B	233 B
Decompression (XB1)	28us	32us	33us	27us

Some concrete examples

	Scramble	Idle	Walk	Cinematic
Num. key frames	24	80	38	2291
Num. animated tracks	**56 (11%)**	**77 (15%)**	**82 (16%)**	**87 (16%)**
Raw size	147 KB	489 KB	232 KB	14498 KB
Compressed size	7 KB	15 KB	12 KB	521 KB
Compression ratio	20 : 1	32 : 1	19 : 1	28 : 1
Avg key frame size	314 B	194 B	325 B	233 B
Decompression (XB1)	28us	32us	33us	27us

Some concrete examples

	Scramble	Idle	Walk	Cinematic
Num. key frames	24	80	38	2291
Num. animated tracks	56 (11%)	77 (15%)	82 (16%)	87 (16%)
Raw size	147 KB	489 KB	232 KB	14498 KB
Compressed size	7 KB	15 KB	12 KB	521 KB
Compression ratio	**20 : 1**	**32 : 1**	**19 : 1**	**28 : 1**
Avg key frame size	314 B	194 B	325 B	233 B
Decompression (XB1)	28us	32us	33us	27us

Some concrete examples

	Scramble	Idle	Walk	Cinematic
Num. key frames	24	80	38	2291
Num. animated tracks	56 (11%)	77 (15%)	82 (16%)	87 (16%)
Raw size	147 KB	489 KB	232 KB	14498 KB
Compressed size	7 KB	15 KB	12 KB	521 KB
Compression ratio	20 : 1	32 : 1	19 : 1	28 : 1
Avg key frame size	**314 B**	**194 B**	**325 B**	**233 B**
Decompression (XB1)	28us	32us	33us	27us

Some concrete examples

	Scramble	Idle	Walk	Cinematic
Num. key frames	24	80	38	2291
Num. animated tracks	56 (11%)	77 (15%)	82 (16%)	87 (16%)
Raw size	147 KB	489 KB	232 KB	14498 KB
Compressed size	7 KB	15 KB	12 KB	521 KB
Compression ratio	20 : 1	32 : 1	19 : 1	28 : 1
Avg key frame size	314 B	194 B	325 B	233 B
Decompression (XB1)	**28us**	**32us**	**33us**	**27us**

Conclusion

- Sweet spot
  - Very fast decompression
  - Reasonably compact
  - High accuracy
  - Future proof

Conclusion

- Versatile
  - Works out of the box
  - Nothing to tweak
  - No need for fallback alternative

Conclusion

- Simple
  - Implemented in 20-25 days
  - No maintenance
  - Easy to build on and improve

Questions?

- **Blog:** http://nfrechette.github.io/