More Thoughts on Time

- Operators
    a. We've been asserting that you can decompose editorial documents into operator graphs
    b. Nick has started doing some nice diagramming of what those operators could look like
    c. ...but have been wondering how that impacted spatial domains
        - What this is referring to is that graphs in practice generally mix spatial transformation with temporal one
        - For example: an OTIO file that composes a series of USD animation caches
        - Or a presto session with several shots open
        - Called horvath to discuss, since he's built a number of these systems and since done time in other industries
    d. Horvath: "craig" notation for spatial transformation from robotics
        - The set of problems you have to solve for spatial transformations is isomorphic to the temporal domain…
        - ...and to other domains (even the spatial domain isn't really the spatial domain but a parameters space like transform or rotation vectors)
        - ...so the dimensions you're passing through your operator graph can be designed and operators can be intermixed
        - Domains that editorial is likely interested in:
            - Spatial, Screen coordinates (like resolution and placement)
            - Spatial, 3d
            - Color space
            - Temporal
        - "craig" notation for transformations from robotics are used for transformations of spatial parameters, but apply equally well to these temporal coordinate frames.
            - Example: local_T_world_matrix * world_T_screen_matrix

Concatenating link transformations

Once the link frames have been defined and the corresponding link pa found, developing the kinematic equations is straightforward. From the valu link parameters, the individual link-transformation matrices can be compute the link transformations can be multiplied together to find the single transfo that relates frame $\{N\}$ to frame $\{0\}$:

$$^{0}_{N}T = {}^{0}_{1}T\,{}^{1}_{2}T\,{}^{2}_{3}T \ldots {}^{N-1}_{N}T.$$

This transformation, $^{0}_{N}T$, will be a function of all n joint variables. If the joint-position sensors are queried, the Cartesian position and orientation o link can be computed by $^{0}_{N}T$.

-
- http://www.mech.sharif.ir/c/document_library/get_file?uuid=5a4bb247-1430-4e46-942c-d692dead831f&groupId=14040 (page 75)

e. We can describe a general graph that can encode a number of different dimensional transformations…
   - ...but only implement the ones we know we need.
   - That leaves us room to use the same notation and description for the event where we need to bring additional dimensions into the model

f. Note that this is not advocating for explicit operator objects in OTIO. Meaning you can't query for a particular operator from inside a clip as a discrete object (the way you can in a presto, Zeno, or Maya environment) but rather the operators describe the algorithmic behavior of the high level entities that compose OTIO structures

g. If we are asking the question of how or if some metadata should be formalized into OTIO data model, we have criteria for what we loft out as follows:
   - If we can say that every object needs to be able to specify an operator on a domain,
   - And that operator needs parameters that exist in a clear coordinate space,
   - Then we can loft the parameters into an operator in a specific place in the object's operator chain (for example, taking the CDL data into a color space operator into the clip's operator chain) so that its arguments can be clearly documented and linked to the explicit operator graph in OTIO, and its parameters are specified to be defined in a specific coordinate system.

h. That leaves the specification of how we define the operators that compose OTIO objects as a next problem; Nick has been working on this in the keynote document.

i. Op { Domain_i { Parameters, Inputs, Outputs } }
   - From an external POV, the Domains are independently specified

        - From a specificational POV, the outputs may be computed internally using any of the parameters and inputs, or any of the op's other domain's parameters and inputs
        - Within the parameters, inputs, and outputs, each of these must be associated with an embedding coordinate frame. The parameters and inputs operate within a local coordinate frame.
- Representation of time (continuous vs discrete)
    a. Arguments for Continuous
        - Animation, interpolation, mechanical devices and simulation are all encoded and evaluated in a continuous space
        - Equivalence of algebraic operators
        - Equivalence of transformed graphs of algebraic operators
            - compiler technology for transformation of a graph of operators from one format to another, with identical output, for example to translate a format from one to another configuration for different user interfaces or applications
            - "Raycasting for answering questions about composed graphs"
        - Lossless transformation between coordinate frames (not necessarily invertible)
        - User interface: you don't need to manipulate a large rational number to transform between rates, the rate will stay meaningful if the value is allowed to be a floating point numbers
    b. Arguments for Discrete
        - Largely practical not mathematical; that is not a negative value judgement
        - Traditional media encodings have integer frame numbers to deal with, both for video and audio
        - Editors, animators, and artists typically think in terms of integer frame numbers
        - It is possible using integer rational numbers to build precise rationals
        - Easier to reverse infinitesimal operations
        - Precision over very long document life times (uint64_t vs double)
    c. Logical argument for the "third way"
        - We both want to consume and capture data in the real world coordinate space of time, in which time is exogenously defined and has the properties of a continuous monotonically increasing normed vector space (because of the classical laws of thermodynamics, which assert that the "arrow of time" points in one direction).
        - Because we want our internal coordinate systems to fit into these exogenous coordinate frames, they must be resolvable into the exogenous real world physical time coordinate space.
        - Time is a continuous monotonically increasing normed vector space

- projecting a discrete time domain value into the continuous playback space implies that a discrete time point is a continuous time range: [sample_start, sample_start + sample_duration)
- discrete time points can be transformed by continuous time transformations
- We know that sampling functions transform from instantaneous time coordinates into discrete ones.
- There is room for both in OTIO, without also forcing a split in the math
- Proposal: have both a discrete and a continuous encoding for time coordinates, but have all the transformations be continuous.
    - Continuous coordinate representation: Double/ unit64/unit64
    - Discrete: uint64/ unit64/uint64
- Opentime needs to have a representation of a discrete point in time as well as a continuous one, and sampling functions for going from one domain to the other.
- Each otio.core.Item should grow a sampling function parameter, which specifies how to map discrete samples to continuous time points and ranges. For example, by snapping.
- Proposal:
    - Change:
        - RationalTime becomes ContinuousTimeCoordinate -> double / unsigned int/unsigned int
            - The components of ContinuousTimeCoordinate are a Value and a Unit (formerly “rate”).
            - Rate becomes “Unit” -- clarifies that the coordinates’ unit is not specifying anything about the rate of the data it may be associated with, only providing a unit for the value coordinate in the coordinate space.
            - Unit is defined to be a positive integer rational multiple of seconds, following typical definitions of units in the real world. 1.0:1/1 means a value of 1 measured on a 1 second unit. 3.2:1/24 means a value of 3.2 measured on a 1/24th of a second unit, or 3.2/24s to convert to seconds.
    - New:
        - Introduce TimeCoordinateValue and TimeCoordinateUnit types for basis compiler type checking
        - DiscreteTimeCoordinate -> signed int over unsigned int / unsigned int.
            - The components of a DiscreteTimeCoordinate are also Value and Unit, embodied by

TimeCoordinateValue and TimeCoordinateUnit classes/typedefs
                - Unit is described in the same way as for continuous time coordinates.
            - SamplingFunction :: ContinuousTimeCoordinate | DiscreteTimeCoordinate -> ContinuousTimeCoordinate | DiscreteTimeCoordinate
                - The role of the sampling function is to conform time transforms into the correct domain of the clip. For example, if a clip is meant to be sampled at 24 hz, the sampling function can perform a floor at 1/24th of a second intervals.
            - Operations that manipulate discrete time may need a sampling policy.
        - Question:
            - Does TimeRange need a discrete/continuous variant?
    d. Worked Example
        - Conditions:
            - Lets say we have a clip with a sampling function of floor()
            - The available_range is [0.0, 40.0) (@ 24/1 rate)
            - It has a source range of [10, 20)
            - It also has a Linear Speed Warp with a value of 2 (play 2x as quickly) @TODO: need a better nomenclature to distinguish between changing temporal density and sample rate
            - After the Speed warp, it has a resample operator that samples the clip with floor at 24
        - If we want to compute the duration of the output space of the clip:
            - We compute the trimmed range in the local space: [10.0 , 20.0) (@24/1 rate)
            - Pass it through the sampling function: [10, 20) @(24/1)
            - Pass it through the speed warp effect: [10, 20) @(48/1)
            - Resample it back down to 24: [10, 16) @(24/1)
    e. Next steps
        - Review opentime in light of ^^ ideas
        - Rebuild as a C-library rather than as a C++ library
- Media Reference
    a. Either:
        - Constrained to be a monotonically increasing coordinate space (current thing)
            - If we go this way, we need to include this in the specification
            - To reverse the order, you'd need to do that with a clip that has a reverse effect

            - We don't have an example that breaks this constraint, so we should start with this constraint until a real world use case need arises that pushes out of this.
        - Or: media references can optionally describe a sampling function that maps the media space to the files on the disk.
- Clarification on Time Coordinate
    a. A time coordinate of a time point in OpenTime is a RationalTime (currently), which we define as a value and rate
        - A better description of the rate component is that it is the units for the value component. A unit cannot be negative, for example there aren't nega-millimeters.
        -
    b. Negative vs Positive Rate:
        - You can't have a negative rate, because you cannot have a negative unit. You can have a negative value, however.
        - We assert that ultimately time is a monotonic increasing normed vector space
        - This means that time always advances
        - Therefore, reversing the rate of playback is reversing the frame mapping from media to local space, but NOT making the playback rate negative.
        - This also means that we can encode rate as a uint64/uint64, because sign does not factor into it.

1. Finish out this doc:
    a. Worked example of the hierarchy traversal between continuous and discrete coordinate frames
    b. Review keynote and make sure this document+the paper is a superset of the ideas
2. Discuss with Josh/Eric
3. Redesign components of opentime based on this document
4. Reevaluate OpenTimelineIO / schema designs
    a. Look at the proposal from the closed PR: https://github.com/ssteinbach/OpenTimelineIO/blob/impl_time_scopes/docs/tutorials/time-scopes.md

- Goals:
    - This document
        - Describe a temporal algebra for consistently evaluating temporal coordinate frames
        - Describe extensions to the OTIO data model for providing enough information to evaluate
    - OpenTime and OpenTimelineIO
        - The domain of the libraries
        - Intended use cases
        - Manner of integration
        - Structural Details

- Architecture: OpenTime and OpenTimelineIO
    - The domain
        - File formats
        - Computations with time
        - Compositions of time sampled and continuous data
        - Evaluations of a composition at a point in time
        - Establish common nomenclature for common operations
        - Establish precise algebraic descriptions of common operations
    - Intended use cases
        - Translation of track based composition between applications
        - Reduction to compositional hierarchies
        - Flattening of compositions to edit decision lists, and atomic operations
        - Future-proof vendor independent archiving of compositions
    - Describe methods of integration
        - When and how should the OTIO C++ libraries be integrated?
        - What properly belongs in the OTIO Python implementation?
        - What should be considered for inclusion in the core distribution?
    - Structural details
        - The exogenous Root Metric Space
            - For time, space, pixels, and color
            - Relationship to discrete and continuous sampling and measurement
        - The innate correspondence and generalization of coordinate system hierarchies familiar to 3d computer graphics across the domains of time, space, pixels, and color
            - The existence of multiple domains in a single composition

            - The resulting need for a formal description of the domains referenced within a composition
        - The OpenTimelineIO file, within the Root Metric Space
            - The Products or Targets of a single OTIO composition
                - The need, and characteristics of a default product
            - The Presentation Domains of a Product
                - The binding of products to the domains
                - THe need, and characteristics of a default domain
            - The compositional hierarchy of an OTIO composition
                - The reason the hierarchy is in reference to the domains
        - The OpenTime and OpenTimelineIO data structures and operators

Goal Products

- TDD
    - Implementation in OTIO
- White paper on OTIO
- Paper on Time
- Keynote
- USD Integration Conversation

What we have:

- Keynote presentation
- Paper
- Github PR on a new API design
- This document
- Random notes on github issues

Ordering of what we have:

- Ideas
    - Paper (1st)
    - Keynote (2nd, builds on paper)
    - This doc (3rd, builds on keynote)
- API Design
    - Github PR
    - Random notes on github issues

Todo:

- Converge everything into this document, before we fan the content out into our goals
    - Paper
    - Keynote

TDD Lens

- [ ] Strawman outline of a tdd
- [ ] Read the keynote and harvest tdd related seeming things
- [ ] Read the paper and harvest tdd related seeming things

Operator Graph Lens

- Linear algebra
    - The transformations of time are homomorphic to transformations on space
    - for connecting coordinate frames to each other
        - Craig notation
    - Transform coordinate frames locally
    - Operators don't need to be explicitly implemented, but are useful for modelling
- Operator graphs can be multi-modal across domains
    - Operator model clarifies what should be a parameter on the operator, and what should be an input or output of the operator
        - Notation for operators
    - We can mix video/audio/time/color operator graphs
    - 3d/spatial /time graphs
    - ...still have consistent transformations
- Equivalency of graphs,
    - Transform file formats
    - modeling of lossiness during such transformation
    - Reduction of graphs at specific time points

Discrete and Continuous Time Lens

Why is Time So Dang Difficult

- Seems trivial
- Cannot directly perceive it
- We tend to manipulate most spatial dimensions in fairly continuous and linear ways, but time is manipulated in discrete and non-linear ways: https://youtu.be/0DLED7krHwU?t=20, https://www.youtube.com/watch?v=icUyG8ZdX04
- The mix of continuous and discrete math that we want to use
    - Discrete:
        - Frames
        - Samples
    - Continuous:
        - Physics

        - Math
- Still has lots of nested coordinate frames
- Demands pedantic precision about non-intuitive concepts like “is my time interval inclusive or exclusive”
- Typically the methods for working with it are ad-hoc, based on physical implementations from antiquity (film reels, NTSC rates, audio sampling rates, etc)

Potential Background Research Areas:

- Midi literature
- Trackers / mixing audio rates inside a project
- std::chrono
- https://www.publichealth.pitt.edu/Portals/0/BIOSTAT/ETD/Pleis_John_dissert_Aug2018.pdf?ver=2018-10-09-142922-260 (“mixing continuous and discrete data”)
- https://www.youtube.com/watch?v=cz4nPSA9rlc

Discrete Vs Continuous time encoding/representation

- Discrete representation
    - assumes that quantization has already happened before the document was serialized (e.g. frames, pixels, audio samples). Quantization is identified as a fundamental quality of the encoding.
    - Mixing quantization rates is ill defined
        - This fundamental quality has defined many aspects of broadcast engineering from the very beginning, for example, media has been supplied at 24fps, yet displayed at 29.97fps. Where we find ourselves today is largely a result of the slow co-evolution of infrastructure and assumptions to sweep this fundamental fault under the carpet.
        - Example:
            - Two Clips are encoded at 30hz, each 0.5s in duration, audio is at 96khz
            - Playback is at 25hz
            - This means that encoded are 15 frames of data before the cutpoint, while the point in time 0.5s is not aligned with a frame boundary at 25hz
            - ...because the audio is at a higher framerate, 0.5s does align with a frame boundary in the higher rate audio
            - If the rounding strategy is to use the floor() function, then
    - The playback system has an explicit rate endogenous to play back the composition. The endogenous rates include image presentation rate, pixel resolution, color space, audio DAC rate and sample bit depth.
- Continuous representation
    - Assumes that quantization happens between interpretation of the document and the playback system. Quantization is effectively on demand. The playback

system needs a sample at a certain time, the sample must be rendered in order to be available.
    - Not all values are representable by floating point representations, which are discrete themselves.
        - Potential for error accumulation due to precision in digital implementations of floating point math
    - Can compose multiple media rates, because all time is effectively encoded in continuous seconds, no question of sampling drift
        - The intention of an editing operation often involves exact discrete alignment to conform to a frame boundary or other discrete event in a system. Whether the conform to the boundary occurs before or after the operation is also a significant factor.
            - Can we design an example that demonstrates this?
                - A flash, occurring at rate A needs to occur on the same resolved frame as another visual cue in media occuring at rate B. The issue is that the flash must not occur one frame early nor one frame late.
    - The core issue of conforming different rates remains
        - What is the governing rate?
        - What happens if more samples are available than the conform rate?
        - What happens if too few samples are available?
- Explicit in-document quantizations operators
    - Can encode both discrete and continuous time values
    - Has explicit transformations from continuous to discrete and vice versa
    - Allows for more forward looking exotic time encodings without sacrificing the precision of discrete time where desired
    - Also allows for mixing discrete time rates by making explicit where and how to transform from one coordinate system to another
        - Can we show an example where this addresses previous issues?
    - Algebraic rearrangement of operations can be used to minimize precision loss from quantization

TDD - OpenTimelineIO

- Summary
    - What is OTIO
        - A narrowly scoped collection of easy to use, high performance foundational types, algorithms, and a file format; written in C++

        - A broadly scoped collection of file format adaptors, independent of the core, community contributed and maintained, written in Python for maximum accessibility
    - Libraries
        - OpenTime
            - Foundational types
            - Algebra
            - Units
        - OpenTimelineIO
            - Data model
            - Integration with OpenTime
            - Coordinate systems
            - Serialization
            - Integration with pipeline/other applications
    - What's in scope
    - What's out of scope
- Features
    - OpenTime
    - OpenTimelineIO
    - Runtime,
    - debugging,
        - Profiling?
        - Metrics?
        - Print formatting?
    - authoring features
        - operators?
- Architecture
    - Objects? Handles? POD?
    - Conventions for calling, pass by reference? Etc
- Components
    - OpenTime
        - C++ Library
        - Python bindings
        - Test suite
        - Utilities?
        - Etc
    - OpenTimelineIO
        - C++ Library
        - Python bindings
        - Test suite
        - Utilities?
        - Etc
- Style Guide

    - Source File layout
    - Naming Conventions
        - Unit specification
        - Global constants
        - Snake casing and camel casing, pep8, exr irix-ish style
    - File naming
    - Library naming
        - Semantic versioning
    - Namespaces
        - Versioning
    - Deprecation conventions
    -
- Classes
    - OpenTime
        - Foundational Types
            - RationalTime --- legacy compat -- deprecate in point release?
            - ContinuousTimeCoordinate
                - Typedefs for common convenience: e.g. otio::ctcf and otio::ctcd
                - Conversions can be provided for convenience between eg ctcf and ctcd, but implicit conversions are not allowed
                - Value
                    - Signed value as a multiple of unit expressed as a floating point value
                - Unit
                    - Positive value in seconds expressed as a ratio of integers
                - Equality
                    - What equality and compatibility means
            - DiscreteTimeCoordinate
                - Value
                    - Signed value as a multiple of unit expressed as a integral value
                - Unit
                    - Positive value in seconds expressed as a ratio of integers
                - Equality
                    - What equality and compatibility means
            - TimeRange -- legacy compat -- deprecate in point release
                - “Range” is imprecise
            - TimeInterval
                - “Interval” has a mathematical meaning

    - TimeInterval encodes Intervals of time, continuous or discrete
    - Start, Duration
    - Clusivity {start, end}
    - TimeRange is equivalent to TimeInterval that is { in, ex }
    - Sentinel values for Start
        - -inf, +inf
    - Sentinel values for Duration
        - +inf
    - Equality
        - What equality and compatibility means
    - Expression of the algebra
        - Reference chainable
- TimeMapping
    - Non-affine mapping, not necessarily a bijection
    - Collection of intervals mapped to Curves
        - Trivial is that curves are held (flat) curves (99.9% case)
        - Complicated one is that they're mapped to curves (animx)
        - Continuity between intervals is also encoded
        - Also include before the beginning and after the end mappings
    - A TimeMapping whose second derivative is 0 everywhere can be represented by a TimeTransform
    - A TimeMapping whose curves are flat everywhere and is defined for all time is essentially a sampling function.
- // @TODO: TimeMappingGenerator?
    - The algorithm behind the Euclidean rhythm generator can help here
- TimeTransform (functional not value)
    - Homogeneous transform
        - Derivation
    - Composition through concatenation
    - Invertible
    - Origin
    - Algebraic Kind (Invertible or not)
    - Presampled
    - TimeTransforms on continuous values are obvious
    - TimeTransform on discrete time points and intervals require an ability to move from discrete to continuous and back
- TimeMapping X TimeTransform ⇒ TimeMapping

- TimeTransform X TimeTransform ⇒ TimeTransform
- TimeMapping X TimeMapping ⇒ TimeMapping
- TimeTransform X TimeMapping ⇒ TimeMapping
- Is there a projection from TimeMapping to TimeTransform? (out of scope because lossy/unprincipled)
- Coordinate System is an abstract quality of an operator
- TimeTransformer joins two labels in an operator graph such that an expression that directly transforms a coordinate at one label into the coordinate system at the other label
    - Is a label an output of an operator?
    - When we walk between two labels, we can query whether we can convert in both directions, or only from one label or the other, by checking each intervening operator for invertibility
- Operators
    - Coordinate -> operator -> coordinate in output space
    - Every operator has an intrinsic space
    - Is the intrinsic the domain of the operator by definition?
    - Every operator knows how to transform a coordinate input into the intrinsic space of the operator, and present it at the output, transformed into the intrinsic space.
    - Some operators can do the opposite, but not all.
    - Does a time mapping live inside an operator?

We want a frame at 2 seconds in the output space
Output space, 30 fps, frames 0 to 120
[O1] Operator converts 2 seconds to frame number 60 at the output space
[O2] Operator that takes 60, pops it back to 2.0, then steps it down to 24 fps, and yields 48
[O3] Operator ‘local space’ takes the 48th frame relative to the start of the clip and transforms that into media space, yielding frame 96 of media
[O4] The media runs from frame 0 to 999 and we are cutting in the source range which starts at 48 and goes until frame 144 (4 seconds, 96 frames). Time 96 @ 24 is valid and returned.

O1 === “output space” domain { start of intrinsic range, duration } , it knows sampling rate, and transforms seconds to a count { ContinuousTimeCoordinate -> DiscreteTimeCoordinate}
O2 === temporal mapping of a count in “output space” to a continuous representation in output space domain, then takes the continuous and maps it to discrete at 24fps

Is the only way to map one discrete domain to another via moving to continuous?

- no: can be done via dithering or residual distribution, e.g. Björkland’s algorithm

Trade off is that the continuous mapping is O(1) but the discrete mapping is O(N) where N is framerate times the value

- :: DiscreteTimeCoordinate : DiscreteTimeCoordinate

O3 === we map the discrete 24 in “output space” to “media space”

- :: DiscreteTimeCoordinate | ContinuousTimeCoordinate : DiscreteTimeCoordinate

O4 == we map “media space” to “trimmed space”

Operators:

- Quantize Continuous to Discrete
- Project Discrete to Continuous
- TimeCoordinate Transformations (scale, translate, apply a TimeMapping)
  - Discrete
  - Continuous
- Interval Operators (slip, trim, reverse, etc)

TimeMapping

- Time spline
- Step Functions

```
TimeCoordinateDiscrete tc_Ds_local;      food_ds   food_cs
TimeCoordinateContinuous tc_Cs_media;
TimeRangeDiscrete tr_Ds;
TimeRangeContinuous tr_Cs;
TimeMappingContinuous output_M_internal;
TimeTransformDiscrete local_T_media;

Result_Ds = tr_Ds * local_DT_media * media_DM_weirdo;
Result_Cs = C_projection_D(tc_Ds_local * local_D_media * media_D_weirdo,
projection_params);

Result_Cs = C_projection_D(tc_localD_s * local_TD_media * media_MD_weirdo,
projection_params);

DiscreteTimeCoordinate result_Ds_media = trim_D(
        /*value=*/ D_resample_D (
                D_sample_C(request_time_Cs_output) * output_DT_local,
                24
        )
        * local_DT_media,
        /* trimParam = */ source_range_Ds_media
)

// this figures the whole mess out by walking the graph:
result_Ds_media = transform(request_time_Cs_output, clip.output_space, clip.media_space)
output_T_media = build_transform(clip.output_space, clip.media_space)
result_Ds_media = request_time_Cs_output * output_T_media;
```

To support that, if we imagine a graph, and we imagine that we can stick a “ground” probe in at one point in the graph, and “sample” probe in at another point in the graph, we should able to create a temporary Transform object that converts times or intervals from ground to sample, and possibly vice versa

```
Class Transform { coordinate operator() (const coordinate&) {}
        Interval operator() (const interval&) {}
 };

// samples in grounds out
Transform ground_T_sample = { ground_probe, sample_probe };

template<typename From>
Class Quantize<From> {
        value<Discrete> operator()(value<Continuous>)

quantize<operator<Discrete>>
```

Capture
[Media <- generated by the media reference
Authoring <- otio defines a number of “authoring” coordinate systems
Presentation] <- generated by the Timeline object
Observation
|~ *Diegetic ~~ possible that OTIO could reason about this, but would require additional metadata*

- Lining - the especially interesting case is being able to link a script to a clip or media
- Picture in picturing
- bingbong

```
Struct diegetic {};
Struct editorial {};
coordinate =  transformer<diegetic, editorial>(coordinatePrime);
Coordinate = transformer<media, presentation>(coordinatePrime); // dependent on invertibility of transformations, available objects, parenting etc.
Struct clip {};
Struct track {};
Struct nestedTrack{};
Transformer nestedtoclip<nestedTrack<track,clip>>
Co = nestedToClip<clip>
```

C0, transform it through spaces to C1’

            - Canonical named coordinate systems
            - Clark notation for coordinate system concatenation
                - Transform form: a_T_b * b_T_c
                - Mapping form: a_M_b * b_T_c
            - Equality
                - What equality and compatibility means
            - Fully worked example
        - Interval Algebra (Time + Ranges)
            - Allen's interval algebra
            - Concatenation via reference chaining
            - notation
        -
        - Graph of operators
        - Queries on the a graph of operators
- Tests

Reject

```
TimePoint
{
      Coordinate c
      MetricSpace m
}
```

```
Because the MS is a quality of the timepoint's type, and modifying m
would have to simultaneously modify c, which would lead to errors
ultimately
```

functor

```
SamplingFunction<MetricSpace Origin, MetricSpace
Destination>(Coordinate c)
{
      M2 -> M1 (TimePoint) // rejects a non-matching TimePoint
```

```
      M1 -> M2 (TimePoint)
}
```

Accept

```
TimePoint<MetricSpace>
{
      Coordinate c
}
```

OTIO provides a ContinuousMetricSpace, and a set of metric spaces that reflect a regular cadence, such as 24hz. Do we also want to provide by default the duplicate-every-1000th-frame metric space?

Is the sampling function a function, or functor? Function:

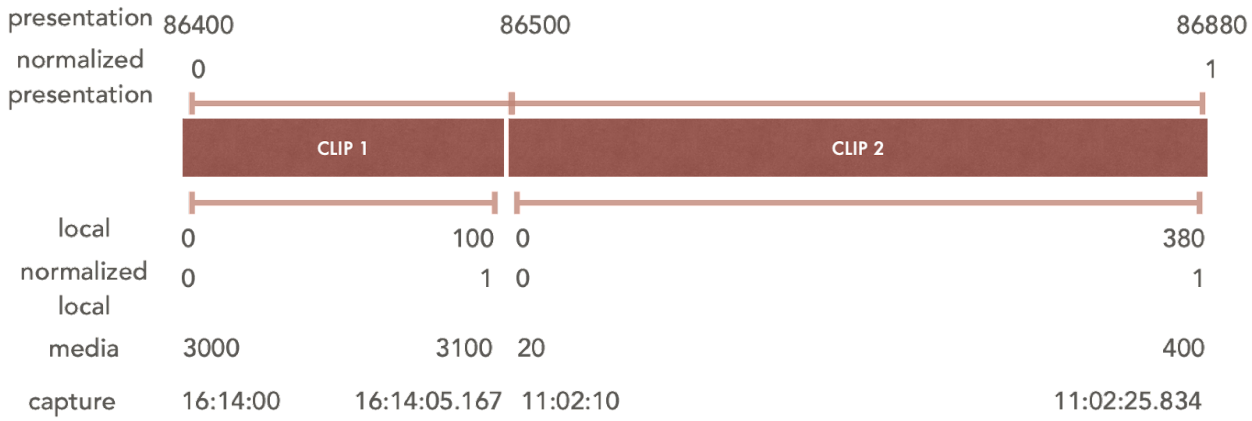
```
template <typename SpaceDst, typename Spacesrc>
 SamplingFunction<SpaceDst>(TimePoint<SpaceSrc> t)

CoordinateSystem<MetricSpace>
{
}
```

Insert graph - given a desire to insert cIns in the middle of c1, in a sequence { c0,c1,c2 } - what graph do we turn the crank on once to generate { c0, c1', cIns, c1'', c2 }

```
Clip
{
      TimePoint origin_TrackCS; // the origin in the Track space
      TimePoint length_ClipCS;
};

Clip c0, c1, c2
Clip cIns

OpIdentity op_idenity
```

```
OpSplit op_split
OpShift op_shift1, op_shift2, op_shift3
OpLength op_length

TimePoint<TrackCS> t_split_tr
TimePoint<ClipCS> t_split_c1

SamplingFunction<TrackCS, ClipCS> track_to_clip

// copy through
op_ident(c0) -> c0'

// transform the split time from the track coordinate system into the
clip's coordinate system
op_xform(t_split_tr, track_to_clip) -> t_split_cs

// split cIns at t_split
op_split(t_split_cs, cIns) -> { cIns' cIns'' }

// get the length of cIns
op_len(cIns) -> t_len_inserted <cIns<MetricSpace>>

// move c1'' into its final position
op_shift(transform(t_len_inserted, TrackCS), c1'') -> { c1_shifted }

// move c2 into its final position
op_shift(transform(t_len_inserted, TrackCS), c2) -> c2_shifted
```

Given a clip on a track, what is the coordinate in clip space corresponding to a coordinate on the track in track space?




```
Root Metric Space (OTIO::Timeline)
{
      "The Magic Cube"
      Continuous
      Origin 0
      Other interesting qualia of a metric space
}

Visual Presentation Domain
{
      Description: "all the visuals"// projector 1 // left eye
      24fps
      Origin 0
      Length 135 minutes
}

/*
Audio Presentation Domain
{
      192khz
}

Fireworks Launcher Domain
{
      10 seconds
}

Animatronic Performer Domain
{}
```

```
Ride Car Domain
{}
*/

Act 3
{
      Embedding: Visual Presentation
      24fps
      Origin 86400
      Length 480
}

Visual Track 1
{
      Embedding: Act 3
      Name: "Location Shoot"
      24fps
      Origin 0
      Length 100000
}

Clip1
{
      Embedding: Visual Track 1
      // note: clip space is the media space
      24fps
      Origin 0
      Length 100
      Media Reference
}

Clip2
{
      Embedding: Visual Track 1
      24fps
      Origin 100
      Length 380
      Media Reference
}


Timeline
{
```

```
      Root Metric Space
      {
            Product
            {
                  Presentation Domain
                  {
                        // Domain specific information
                  }
            }
      }

      Tracks (Timeline hierarchy)
      Metadata
}


Root Metric Space
{
      Movie Product
      {
            Visual Domain (Rec709 primaries, D65, resolution, 24hz)
            Mobile Visual (...)
            Audio Domain (48khz, 16bit float, 7.1 Surround)
            Mobile Audio Domain (48khz, 16bit int, stereo)
      }
      Ridefilm Product
      {
            Visual Domain (Rec2020 primaries, D65, resolution, 120hz)
            Audio Domain (48khz, 16bit float, 100 channels)
      }

      Tracks (::Stack)
            MovieTrack
                  -> (Visual, Audio*)
                  Clip1 X Trim2
                  Clip2
            RideTrack
                  -> (Ridefilm.*)
                  Clip1 X Trim1
}
```

What's the coordinate in the media space of a clip given a coordinate in presentation time?

```
TimePoint t1

// the set of clips that exist at the presentation coordinate
OpFilter(All clips, presentation coordinate) -> Filtered Clips

// each presented frame in each presented clip
For all filtered clips F:
     -> F_frame_inFrameCS


 // @NOTE also need to be able to do this, but we think all the
information we want to provide to answer this question:
// given the list F_frame_inFrameCS, output a new list of compositing
operations such as (Clip 1 frame 17) blend(50%) over (Clip2 frame 8)
--- which obviously isn't this case in the picture
```

very excited that we could express that just now!

5:03

I feel like we've been dancing around that for a while and it clarified itself for us!

5:04

(pushed)

5:06

seriously that time as a polygon thing is lighting up my brain! so cool

> Nice! It's been tickling my brain since we started talking about that slide title (A Calculus of Time, A Time Algebra, ...) I kept wondering if there is actually a Topology of Time, and we just figured it out, there's a Topology not of time, but of the time continuum. So an editorial timeline is a topological construction of intervals into a temporal continuum.
> https://plato.stanford.edu/entries/time/
>
> section 3 seems to make that argument: https://plato.stanford.edu/entries/time/

5:20

"It is also worth asking whether time must be represented by a single line. Perhaps we should take seriously the possibility of time's consisting of multiple time streams, each one of which is isolated from each other, so that every moment of time stands in temporal relations to other moments in its own time stream, but does not bear any temporal relations to any moment from another time stream."

5:21

That seems to be what we have derived

> Section 8 classifies what we are doing in the "Static Theory of Time" bucket, where time can be treated geometrically
>
> > "The universe is spread out in four similar dimensions, which together make up a unified, four-dimensional manifold, appropriately called *spacetime*.
> > Any physical object that is located at different times has a different temporal part for each moment at which it is located.
> > The correct ontology does not change over time, and it always includes objects from every region of spacetime.