Rectangular Mask Payload Metadata within the QuickTime Movie File Format

Format additions

Version 0.9 (Beta)

June 9, 2025

Note: The information contained within this document is preliminary and is subject to change.

Introduction

Decoded video frames are most typically shown in their entirety to the viewer; the video's bounding rectangle never changes. Creators may, however, want to present less than the full encoded frame (e.g., for the effect of a transition between two aspect ratios that establish a sense of era, such as between a widescreen theatrical aspect ratio and a *legacy* 4:3 television aspect ratio). This can be accomplished today by encoding to a fixed resolution and using letterbox and/or pillarbox "black bars" that are still shown on the projection screen or display.

New display technologies offer new creative opportunities, such as treating these black bars as transparent. Such a creative effect could be accomplished if there existed a way to signal a current rectangular limit on what is shown. If that rectangular area changes as video plays, the display system could treat areas beyond the area as transparent. This document does not prescribe the visual treatment a player uses in its interpretation of this rectangular area, also known as a *rectangular mask*. Rather, this document specifies the structure of a timed metadata payload that can indicate a fixed or changing rectangular area extracted from otherwise constant-sized decoded video.

This payload can be carried in metadata items in media samples within QuickTime File Format [QTFF] timed metadata or ISOBMFF [ISOBMFF] multiplexed metadata. Both of these use the '`mebx`' handler type of the '`meta`' track type. The payloads can also occur in fragmented movie files in both QTFF and ISOBMFF. Although this specification is focused on use within a timed metadata track, the construct itself should be applicable elsewhere in the format.

The document begins with motivations and goals of the payload definition. This is followed by details of the format definition itself, including use of it to describe payload design details and key-related information for carrying this masking rectangle.

Overview

Traditionally, video is prepared at a particular resolution (e.g., 1080p, 4K) and later shown on a two-dimensional display, scaling larger or smaller as needed to fit the display area. This typically targets consumer devices such as iPhone, iPad and Mac, displays attached to desktops, and televisions. Video might also be displayed on a very large screen, such as in a cinema, or on a conference presentation stage. Today, it might also be shown as 2D video within the spatial environment established by a spatial computer, such as the Apple Vision Pro.

Those producing the video might prepare the content so it has a different *aspect ratio* (the ratio of width to height) from the display device. For common cases, such as 16:9 (said as "16 by 9") or 4:3, displays may be sized so the video scaled to the aspect ratio doesn't leave any space beyond the video. In other cases, an encoded aspect ratio such as 16:9 might be displayed where extra, unaccounted-for space might abut some edges. This can manifest as black bars at the top and bottom (termed *letterboxing*) or the left and right (termed *pillarboxing*). For traditional 2D displays, any unaccounted-for area up to the display's physical edges needs to be filled, either explicitly (by painting a constant color such as black) or with whatever the device display shows there.

Sometimes the encoded video might need to target a well-known aspect ratio, such as 4:3, whereas the source for encoding is another aspect ratio (e.g., 16:9 or a theatrical aspect ratio). This difference requires the video production to bake in the letterbox or pillarbox treatment.

A more dynamic creative choice, used by some theatrical filmmakers or other creatives, is producing video that changes the active area as playback proceeds. For example, to signal a shift in era, a filmmaker might transition from a modern aspect ratio, like 16:9, to the 4:3 aspect ratio used on TV in the 1950s or 1960s. On traditional displays, the margins around the active area change as the transition proceeds.

For a device such as Apple Vision Pro, there is a new choice: the area beyond the encoded video's active area, up to the encoded edges, can be treated as transparent, with pass-through visibility to the immersed environment. In this case, there is no reason to present the black bars required for more traditional displays.

Signaling such a static or changing bounding area subset of the video needs a mechanism. That mechanism might carry a single rectangle that can be held constant or can change as the video plays. That rectangle could be used to display an indicated subset of each corresponding video frame.

This document describes a structure that indicates a rectangle suitable for a particular resolution and how that is carried in the timed metadata format of the QuickTime File Format [QTFF] and the ISO Base Media Format [ISOBMFF]. It introduces something called a **rectangular mask**.

Note: This document uses the ISOBMFF specification syntax [ISOBMFF] and uses the QTFF term *atom* interchangeably with the ISO term *box*. When describing syntax and constructs, *box* is used. Although these constructs are likely most applicable to timed metadata items, they might prove useful in other parts of the file formats.

Note: The words *may*, *should*, and *shall* are used in the conventional specification sense—that is, respectively, to denote permitted, recommended, or required behaviors.

References

This document references these external specifications:

[QTFF] QuickTime File Format specification

[ISOBMFF] ISO Base Media File Format, or sometimes ISOBMFF, as specified in ISO/IEC 14496-12 (2020)

Motivations and goals of rectangular masks

Characterizing the mask area of an associated video frame

Characterizing features of areas of the video frame may prove useful or important for particular rendering or UI treatments, such as indicating a rectangular area that should be displayed but having a different treatment for any area of the video frame that extends beyond this masking rectangle. Although this document is focused on a masking rectangle, the construct described here is more general and could apply to other purposes, such as indicating an area of interest within the frame.

Suitable for stereo video with or without window violations

Stereo video uses two image buffers, presenting one image to the viewer's left eye and the second image to the viewer's right eye. For the stereo case, a separate rectangular mask for each viewer eye may be desired. This specification also describes how that carriage can be achieved.

With stereo video, a phenomenon can occur where differences between what the two eyes see in emergent elements at the view edges can produce viewing-comfort issues. In theatrical 3D content production, this is sometimes addressed by painting rectangular or even trapezoidal areas at the offending edge so the eyes do not discern a difference. The literature goes into detail about the need to address window violations with techniques such as floating windows. This specification describes how to signal a pair of rectangular masks suitable for corresponding stereo frames. It also describes a different payload that indicates a constrained trapezoid meant to be suitable for indicating where a window violation should be masked.

Separation of mask shape from encoded video sample

The rectangular mask metadata design described here will typically be carried in a timed metadata track (i.e., the '`mebx`' media subtype format described in QTFF and in ISOBMFF).

Although an alternative might have been to include the mask rectangle semantics in a Supplemental Enhancement Information (SEI) message, the use of timed metadata and separation of the rectangle from the encoded video frame offers benefits including:

- Multiple metadata tracks with different masks can be associated with the video track (e.g., for fine-tuning placement).
- New mask metadata can be introduced without needing to rewrite the video.
- Because it is not in the encoded video, the video mask(s) can be delivered to the client without requiring delivery and decode (or even scanning) of the encoded video frames.
- Because it is not in the encoded video, different resolutions of video can be encoded and used with the same rectangular mask metadata. This might be useful in streaming tiers differing by resolution.
- The same metadata sample and its mask can be applied to more than one video sample if the area is the same or deemed sufficiently similar.

While rectangular masks are most likely to be used with timed metadata tracks, the design should be able to be used with static metadata as an item so long as no timing is signaled in the metadata item payload.

Differing aspect ratios

Differing standard-size aspect ratios can be described by using rectangular masks—such as a 16:9 active area within a 4:3 encoded video frame (letterboxing), or a 4:3 active area within a 16:9 encode video frame (pillarboxing)—and transitioning between the two.

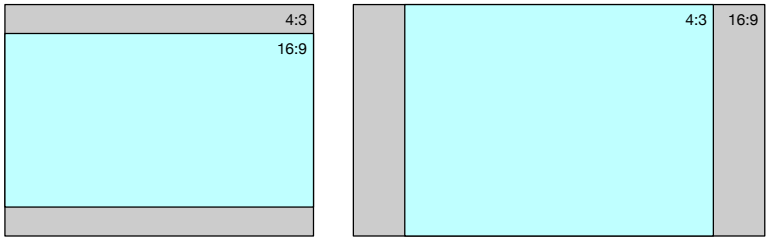

Fig 1. 16:9 within 4:3 and 4:3 within 16:9

Note: Per-frame rectangular mask metadata targets creative cases where the size and location of a portion of the encoded video changes as video plays. If, however, the size of the active area is constant across the video's duration, a traditional *clean aperture* can be used. Or the video can be encoded to a smaller static size without needing a clean aperture. Modern playback takes care to center the encoded video as necessary.

It's probably useful to think of the encoded frame size as a canvas in which a rectangular area can change, exposing a subset of the video. One use might be to introduce a portion of the video at a location on the display and expand the rectangle to show more of the scene, all while keeping the location of elements stable despite more being exposed.

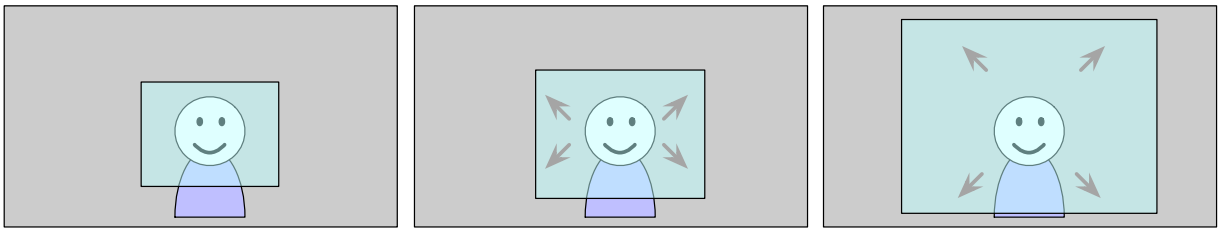

Fig 2. Encoded video with mask being changed toward edges

The shape of the rectangular mask does not need to conform to an established aspect ratio such as 16:9 or 4:3. And the area exposed can be any part of the canvas established by the encoded video dimensions.

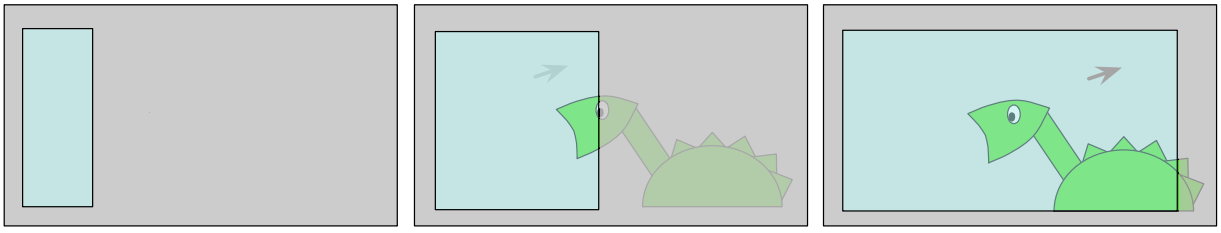

Fig 3. Encoded video as canvas with mask being changed to expose more

Integrating time

Video rectangular mask metadata items are carried as timed metadata track samples. Each metadata sample can correspond to one video sample's presentation time and duration, or it can correspond to more than one video sample. The duration of the mask metadata samples should be the maximum time interval corresponding to whole video frames where the mask metadata is held constant. This metadata interval should be no less than one video frame and it can extend up to the duration of the corresponding video track.

This specification focuses on movie file carriage and does not prescribe rules for derived use cases, such as streaming segmentation, that may need to restrict the time range of metadata samples. Where those derived use cases need to split a long-duration metadata sample, a longer-duration mask metadata sample described here can be duplicated, with the earlier and later samples retimed to account for the overall duration. Such splitting may require consideration of other metadata items in the metadata sample, and their suitability for being duplicated in the same way. Currently, no other metadata requires special treatment when being split and retimed.

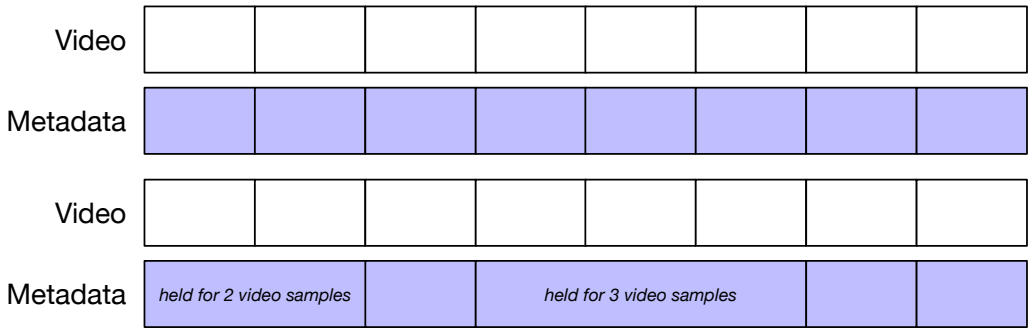

Fig 4. Video and associated timed metadata (1:1 and longer duration)

Because a metadata sample can span multiple video frames in a movie file, it is recommended to have one metadata sample with a mask rectangle that covers all consecutive video frame samples. HTTP Live Streaming (HLS) segmentation may split such a run of consecutive metadata samples to align with segmentation boundaries identified for the corresponding video frames. If there is any change in masks from frame to frame, a new metadata sample run is required at each change. Even if the same mask is possible across a span of video frame times, you should write longer-duration metadata samples in a direct relationship with the corresponding video frame samples.

Because per-frame rectangular metadata (PFRM) usage is tied to a corresponding video sample or samples, it is recommended that the metadata items for PFRM be kept in a timed metadata (mebx coding type) that does not hold other kinds of timed metadata. Additionally, the timed metadata track should carry a 'rndr' track reference so that the PFRM metadata is tightly coordinated with concurrent video frame delivery. Moreover, because the duration of PFRM metadata may extend beyond one video frame, PFRM metadata should be carried in a separate timed metadata track from the track that another, more fine-grained (1:1) or more loosely grained (1:N) timed metadata might use.

Client and production usage

The use of rectangular mask metadata might be different in production and client delivery.

As an example, in a first pass, production might analyze each video frame for any encoded letterbox or pillarbox bands and then produce associated timed metadata samples characterizing the corresponding video frames. The first pass might alternatively use sideband information to generate the timed metadata. Subsequent passes might produce metadata that integrates longer time ranges of already determined mask rectangles, up to the point where it might be constant across the whole video. In this way, the mask metadata described here is both generated, and used as source for generating “optimized” mask metadata.

Client-delivered metadata must exist and be interleaved with the video for each tier resolution in a streaming asset when it applies to the video. The timed mask metadata may be specialized for each tier video resolution, or the timed mask metadata may use a common tier resolution that will be scaled for the tier resolution. Guidance on specializing the mask metadata resolution for each tier or choosing a suitable single resolution to target is not described in this specification. Separate authoring guidelines or other policy can describe that.

Rectangular mask value and metadata carriage

This section describes the structure of the rectangular area and its use as timed metadata.

1. Raster rectangle value payload definition

This section describes the `RasterRectangleValue`, a value to describe a rectangular area contained with a larger, two-dimensional pixel-based area.

1.1. Syntax

```
aligned(8) class RasterRectangleValue {
   unsigned int(16) reference_raster_width;
   unsigned int(16) reference_raster_height;
   unsigned int(16) rectangle_left;
   unsigned int(16) rectangle_width;
   unsigned int(16) rectangle_top;
   unsigned int(16) rectangle_height;
}
```

```
aligned(8) class ExtendedRasterRectangleValue extends RasterRectangleValue
{
   unsigned int(4) left_edge_point_count; // up to 15 edge points along
left edge
   unsigned int(4) right_edge_point_count; // up to 15 edge points along
right edge
   {
      unsigned int(16) inset_x;
      unsigned int(16) inset_y;
   } left_edge_points [left_edge_point_count];
   {
      unsigned int(16) inset_x;
      unsigned int(16) inset_y;
   } right_edge_points [right_edge_point_count];
}
```

2. Semantics

`RasterRectangleValue` is a structure that specifies a pixel-accurate rectangular area with reference to a containing raster resolution (i.e., horizontal pixels by vertical pixels). The value carries both a raster resolution (width and height in pixels) and a contained rectangular area. That rectangle is defined by an origin and size. This geometry value is for general use, and the rectangle applies to the specified resolution but can also be scaled to alternate resolutions. The structure's byte ordering is big endian (in other words, *network byte order*) to be consistent with that of QTFF and ISOBMFF constructs.

- `reference_raster_width` and `reference_raster_height` measure a 2D raster in pixels. This reflects the dimensions of a decompressed pixel buffer or the resolution of an output device. This defines a coordinate system with a left-top origin of 0,0 and a width of `reference_raster_width` and height of `reference_raster_height`.
- The rectangle itself is specified by the four other fields of `RasterRectangleValue`: `rectangle_left`, `rectangle_width`, `rectangle_top` and `rectangle_height`. The interpretation of the coordinates is established by the associated `reference_raster_width` and `reference_raster_height` fields.
  - `rectangle_left` is the horizontal pixel offset from the left of the bounding raster. Because the bounding raster's left edge is at 0 pixels, this is 0 if at the left.
  - `rectangle_width` is the width of the rectangle, from `rectangle_left` to the rectangle's right edge within the bounding raster.
  - `rectangle_top` is the vertical pixel offset from the top of the bounding raster. Because the bounding raster's top edge is at 0 pixels, this is 0 if at the top.
  - `rectangle_height` is the height of the rectangle, from `rectangle_top` to the rectangle's bottom edge within the bounding raster.

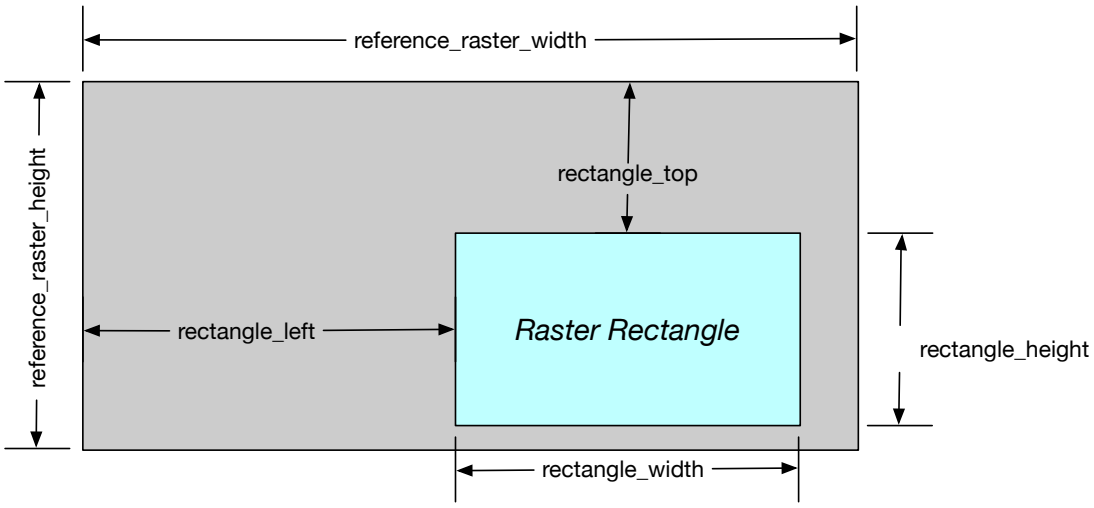

Fig 5. `RasterRectangleValue` rectangle within referenced raster

Note: A `RasterRectangleValue`'s rectangle is tied to a particular resolution indicated by the `reference_raster_width` and `reference_raster_height` fields. If the video frame's resolution matches the value's resolution, the rectangle values can be used directly. If the resolutions differ, the ratios of the current `reference_raster_width` and `reference_raster_height` fields to the other `RasterRectangleValue`'s resolution can be used to scale the origin and size between this and the other rectangle.

To address window violations, a variation of `RasterRectangleValue` is described. Toward that end, Extended`RasterRectangleValue` is introduced. This value is an extension of `RasterRectangleValue`, with additional fields to introduce horizontal insets on the left and right edges.

- The rectangle itself is specified by the four fields of `RasterRectangleValue`: `rectangle_left`, `rectangle_width`, `rectangle_top` and `rectangle_height`. More complex edge geometry is produced by using additional insets using the `left_edge_points[]` and/or `right_edge_points[]` array fields.

  A common case might be a vertical but inset left edge and/or with a vertical but inset right edge. Toward this end, we can introduce for the purpose of this section abstract, or virtual, fields named `top_left_inset`, `bottom_left_inset`, `top_right_inset`, and `bottom_right_inset`. These are not actual fields in `ExtendedRasterRectangleValue` but can be expressed as equivalents of fields that are present.

  - A virtual `top_left_inset` is the horizontal pixel offset from the top-left corner of the specified rectangle toward the right. A value of 0 indicates no offset. This would be

expressed by setting `left_edge_points[0].inset_x` to the virtual `top_left_inset` value and `left_edge_points[0].inset_y` set to 0.

- A virtual `bottom_left_inset` is the horizontal pixel offset from the bottom-left corner of the specified rectangle toward the right. A value of 0 indicates no offset. This would be expressed by setting `left_edge_points[left_edge_point_count-1].inset_x` to the virtual `top_left_inset` value and `left_edge_points[left_edge_point_count-1].inset_y` set to `rectangle_height`.
- A virtual `top_right_inset` is the horizontal pixel offset from the top-right corner of the specified rectangle toward the left. A value of 0 indicates no offset. This would be expressed by setting `right_edge_points[0].inset_x` to the virtual `top_right_inset` value and `right_edge_points[0].inset_y` set to 0.
- A virtual `bottom_right_inset` is the horizontal pixel offset from the bottom-right corner of the specified rectangle toward the left. A value of 0 indicates no offset. This would be expressed by setting `left_edge_points[right_edge_point_count-1].inset_x` to the virtual `bottom_right_inset` value and `left_edge_points[right_edge_point_count-1].inset_y` should be set to `rectangle_height`.

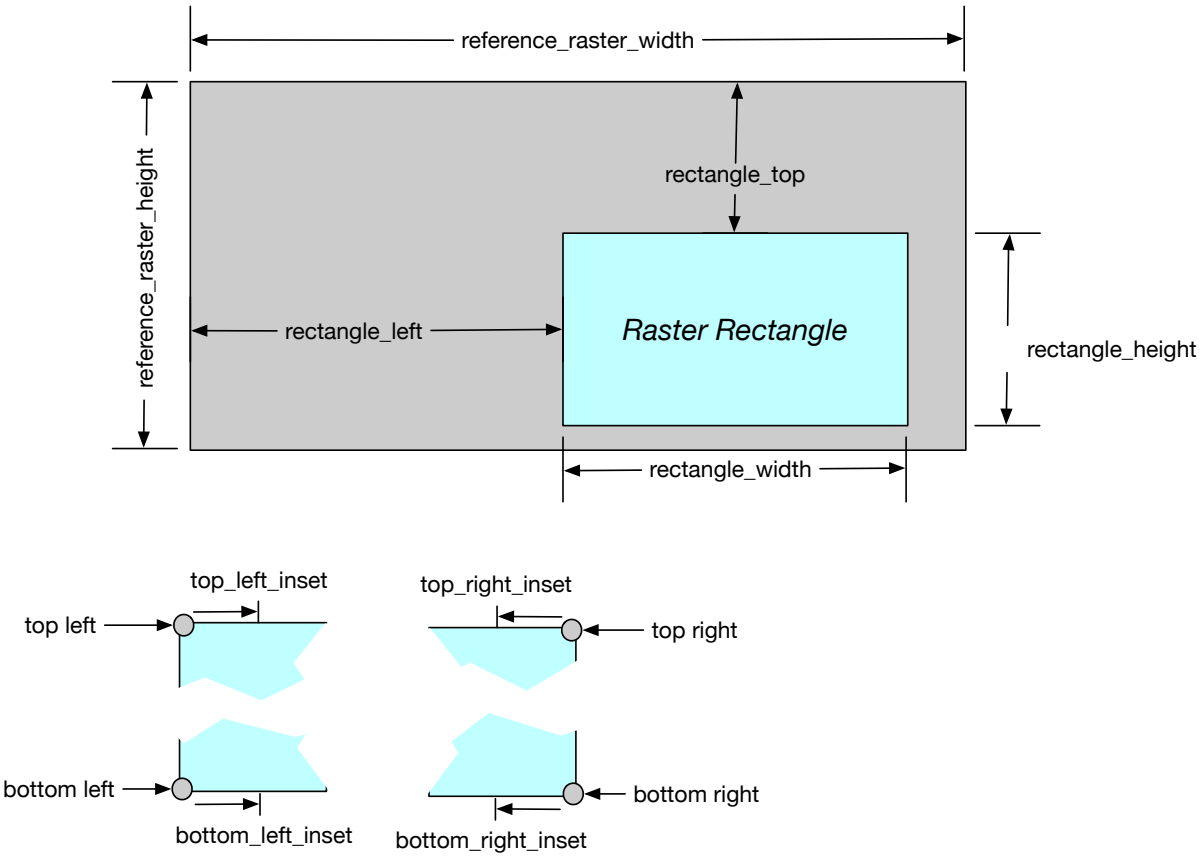

Fig 6. `RasterRectangleValue` rectangle within referenced raster

Although an Extended`RasterRectangleValue` with 0 for all of `inset_x` fields of `left_edge_points[]` elements and `right_edge_points[]` elements is semantically equivalent to `RasterRectangleValue`, the simplest type should be used to describe the necessary geometry.

> **Note:** The misuse of insets might produce self-intersection with the RasterRectangleValue geometry. Such geometry should be avoided, and its use is undefined.

`left_edge_point_count`, `right_edge_point_count`: If a vertical edge has any inset other than 0 across its height, this field should indicate the number of insets on that edge. If there is no inset other than at the top and bottom, `*_edge_point_count` should be 0. Otherwise, there should be that many elements in the `left_edge_points` and `right_edge_points` arrays. If both count fields are set to zero (0), `RasterRectangleValue` can be used instead of `ExtendedRasterRectangleValue`. `ExtendedRasterRectangleValue`

`left_edge_point_count` indicates the count of needed insets for the left edge. `right_edge_point_count` indicates the count of needed insets for the right edge.

`left_edge_points`, `right_edge_points`: An array of elements indicating the offset along the respective edge of the `ExtendedRasterRectangleValue`. Each element is a structure of two fields:

- `inset_x`: An unsigned offset (0 or positive) from the respective edge toward the other edge. For `left_edge_points`, the inset is toward the right. For `right_edge_points`, the inset is toward the left. These inset values shall never be as large as `rectangle_width`.
- `inset_y`: An unsigned offset (0 or positive) from the top edge toward the bottom edge. All `inset_y` values are measured from the top edge. Inset values shall be within the range of 0 to `rectangle_height`. They are not delta values. Subsequent elements shall be monotonically increasing (i.e., never regress or even be coincident with the immediately previous element's `inset_y`).

The use of left_edge_point_count and left_edge_points allows for both complex contours and a simple vertical line. A line from the top-left corner to the bottom-left corner could be described with these field values:

```
left_edge_point_count = 2
left_edge_points[0].inset_x = 0, left_edge_point[0].inset_y = 0
left_edge_points[1].inset_x = 0, left_edge_point[1].inset_y = rectangle_height.
```

A line from the top-right corner to the bottom-right corner, coincident with the right edge, could be described with these field values:

```
right_edge_point_count = 2
right_edge_points[0].inset_x = 0, right_edge_point[0].inset_y = 0
right_edge_points[1].inset_x = 0, right_edge_point[1].inset_y = rectangle_height.
```

Although using `*_edge_points` arrays is possible in these cases, writers should instead use the edge already specified by the `RasterRectangleValue`'s base rectangle-related fields,

set `left_edge_point_count` or `right_edge_point_count` to 0, and not include any elements. Only use `*_edge_points` if the geometry differs from an edge.

1.3. Policy for working with RasterRectangleValues mathematically

The client reader receiving and processing mask metadata should be prepared for a number of situations in determining what is within the specified masked area and, by implication, what is outside the specified masked area.

- If the current image buffer has the same resolution as that specified in the `RasterRectangleValue` resolution (i.e., `reference_raster_width` and `reference_raster_height`), the coordinates of the embedded rectangle (i.e., `rectangle_left`, `rectangle_top`, `rectangle_width`, and `rectangle_height`) can be used directly.
- If the current image buffer resolution does not match the `RasterRectangleValue` resolution, the client should scale the rectangle geometry (`_left`, `_top`, `_width`, and `_height`) between the metadata item's `RasterRectangleValue` resolution and the current image buffer resolution.
- The width and/or height may collapse to 0 and make the geometry empty, or they can be specified as 0. This is valid and should be interpreted as the equivalent of an alpha-based image having no visible pixels, because all alpha is transparent. For rectangular and trapezoidal masks, the area within the geometry should be treated as opaque and pixels outside the mask as transparent. The reader performing rendering can treat the edge of the geometry as a hard cut between opaque and fully transparent, with no obligation (or affordance) for indicating the edge is antialiased or feathered, or otherwise specially treated. If the rectangle extends beyond the raster, all pixels beyond the raster should be clipped (ignored). Only the image buffer within the raster dimensions contributes pixels to be displayed.

```
Given: image_buffer_width // width of the current image buffer
     image_buffer_height // height of the current image buffer
     current_raster_rectangle // RasterRectangleValue read from
metadata item
     mask_left, mask_top, mask_width, mask_height // to be
calculated


If ((image_buffer_width ==
current_raster_rectangle.reference_raster_width) &&
image_buffer_height ==
current_raster_rectangle.reference_raster_height)) {
     mask_left = current_raster_rectangle.rectangle_left;
```

```
     mask_top = current_raster_rectangle.rectangle_top;
     mask_width = current_raster_rectangle.rectangle_width;
     mask_height = current_raster_rectangle.rectangle_height;
} else { // scale the geometry, avoiding integer division
collapsing to zero (0)
     mask_left = (image_buffer_width /
current_raster_rectangle.reference_raster_width) *
current_raster_rectangle.rectangle_left;
     mask_top = (image_buffer_height /
current_raster_rectangle.reference_raster_height) *
current_raster_rectangle.rectangle_top;
     mask_width = (image_buffer_width /
current_raster_rectangle.reference_raster_width) *
current_raster_rectangle.rectangle_width;
     mask_height = (image_buffer_height /
current_raster_rectangle.reference_raster_height) *
current_raster_rectangle.rectangle_height;
}

// use mask_left, mask_top, mask_width, mask_height as shape
//     for pixels to include (or pass through) from the image
buffer
```

1.4. Carriage as timed metadata

The `RasterRectangleValue` and the ExtendedRasterRectangleValue derived from it are each a big-endian structure.

When carried in ‘`mebx`’ timed metadata [QTFF, ISOBMFF], the *per-frame rectangular mask* metadata is carried as either a RasterRectangleValue or an Extended`RasterRectangleValue` item using one of the following keys and value definitions:

Keyspace	Key	Data type[1]	Well-known data type[2]
mdta	com.apple.quicktime.video.display-mask-rect.mono	`RasterRectangleValue`[3]	84
mdta	com.apple.quicktime.video.display-mask-rect.stereo-left	`ExtendedRasterRectangleValue`[4]	85
mdta	com.apple.quicktime.video.display-mask-rect.stereo-right	`ExtendedRasterRectangleValue`[4]	85

Note[1]: *BE* refers to *big-endian*.
Note[2]: ISOBMFF does not currently support signaling data types like QTFF does.
Note[3]: For mono, a RasterRectangleValue is used because there is no expectation that additional edge geometry is needed.
Note[4]: ExtendedRasterRectangleValue may be limited in size to that of RasterRectangleValue if no additional edge geometry is specified.

Each such metadata sample is associated with the presentation time of one or more video frames and describes the portion of the decoded video that should be presented. All pixels outside the rectangle described by `RasterRectangleValue` can be treated as not available. That can be interpreted as having transparent alpha, or in some other way appropriate for the player. This document does not prescribe a treatment.

If the metadata-described area is empty, no corresponding image buffer pixels are produced, and so there's nothing visible.

If no mask-related metadata item is present for the corresponding video frame, the video frame should be shown. The only way to hide it is to have an empty mask geometry described by one of the mask-related metadata keys. Note: This approach helps prevent a production issue in which no timed metadata track is multiplexed with the encoded video track, and nothing is shown. To hide the frame's pixels, use a suitable mask.

> **Note:** There may be future alternatives to per-frame rectangular mask metadata, offering new approaches for identifying areas to treat as transparent. This specification does not prescribe nor limit such additions. In general, the expectation is that those mechanisms will have their own signaling and will be used independently of this metadata approach. Mixing of mechanisms described in this document and future mechanisms will be associated with guidance, but the expectation is that only one mechanism should be used.

Conclusion

The rectangular area defined with `RasterRectangleValue` and the documented timed metadata keys can be used to limit the portion of decoded video that should be presented to the viewer. The current `RasterRectangleValue` structure may be used in future versions of this specification, for other cases in which describing a rectangular area is useful. Though initially intended to be carried as a metadata item within a '`mebx`' timed metadata track, the construct may be incorporated into other parts of the QuickTime File Format. The design should be applicable to ISOBMFF-derived formats if that proves useful.

Document Revision History

Date	Revision	Notes
2025-06-09	0.9	First version