MVEX

Minimal Video Extension to X
Version 6.0

Todd Brunhoff

Tektronix, Inc.
Tektronix Research Laboratories

1. Overview

The spirit of this extension is to provide an X interface to the generally interesting aspects of displaying live video in windows, capturing graphics from windows and converting them to a video signal. Based on an earlier extension called VEX, this extension endevors to provide the minimal support needed for integrating video into the window system.

For applications that need access to the digitized video pixels, this extension allows normal pixmaps or windows with core visual classes to hold video, which in turn allows read and write access to the pixels via the core protocol or other extensions such as the X Image Extension. For hardware that does not actually digitize any pixels, MVEX introduces two new visual classes: `VideoColor` and `VideoGray`. Their purpose is to express the visual aspects of a visible but untouchable video picture, hence drawing on a window with one of these visual classes has no effect and the pixels are undefined.

Today, video input and output hardware has complex limitations and capabilities; the capabilities do not always scale in expected ways, nor do the limitations always make sense. The MVEX extension can provide large amounts of information about these limitations and capabilities. It is intended that the client, through the use of convenience routines in the MVEXlib implementation, will be able to easily determine what it can and can't do without the overhead of a round-trip request to the server.

2. Terminology

Full-motion video

The display of video on the screen at a rate at least as fast as that defined by the encoded video signal. For example NTSC is defined as (roughly) 30 frames per second and 60 fields per second.

Opaque pixels

Some video input hardware uses a single frame buffer to hold both video pixels and graphic pixels. If the video pixels are stored in a form that does not match a core visual class, then the meaning of those pixels, from a client perspective, is said to be opaque.

Still video

A single, static frame acquired from a video signal.

Video digitization

A process that converts a video image from an encoded signal into a digital form having a supported depth and format.

Video input

The term *video input* in general refers to an incoming signal that is decoded or digitized to produce a picture on the workstation display. In this document, video input refers to the digitizing or decoding hardware, and it is identified by a VIDEOIN id. Logically, a client will connect an output device, such as a video disk player, to a video input using the ChangeConnectivity request, and then use `RenderVideo` to direct in which drawable the picture will appear.

Video input region (VIR)

This is defined as the subset of pixels in a VW or a pixmap that is actually being updated by a `RenderVideo` request.

Video output

The term *video output* in general refers to signals produced by encoding workstation graphics. In this document, video output refers to the encoding hardware, and it is identified by a VIDEOOUT id. Logically, a client will connect an input device, such as a video tape recorder, to a video output using the ChangeConnectivity request, and then use `CaptureGraphics` to direct from which drawable the graphics will be captured.

Video output region (VOR)

This is defined as the set of pixels on the screen or in a pixmap that are being encoded by a `CaptureGraphics` request, whether it is accomplished by reading pixels from a frame buffer or by analog conversion.

Video window (VW)

A video window is defined as any window actively displaying video or one that has the potential to display nothing but video. Specifically,

- a window having a visual class `VideoColor` or `VideoGray`.
- a window with a core visual class, and actively servicing a `RenderVideo` request.

3. Visuals

There are three important classes of video input hardware. Each class produces a video picture on the screen using a different method, and the methods affects the visual class used to represent the window or pixmap that contains it. MVEX attempts to recognize all three classes:

- Core visual pixels. Many instances of hardware choose to represent a video picture in a frame buffer that is no different than most graphic frame buffers. In this case nothing new need be added, because one of the core visual classes is sufficient.
- Inaccessible pixels. Some low-cost video input hardware is contained in a separate subsystem where either pixels are never digitized, or if they are, the pixels are available only at low serial line speeds. The analog video picture is merged with the workstation graphics analog signal such that the picture is updated very quickly. Therefore, this hardware will continue to be reasonable as long as video is not a standard option for every X display. For this, MVEX introduces two new visual class, `VideoGray` and `VideoColor`, whose semantics essentially describe a write-only visual: video pixel values cannot be read. This implies that pixmap access for these visual types will never be offered.
- Opaque pixels. A few instances of hardware choose to represent digitized pixels as an encoded color difference, such as YUV; Y,R-Y,B-Y; or YIQ. This encoding is more compact than RGB, but cannot be represented with any of the core visual classes. Rather than invent more new color models and visual classes, MVEX allows the use of whatever core visual class matches the frame buffer’s "normal" mode, but pixels that are modified by `RenderVideo` are opaque. This implies that while the pixels can be read, their meaning is undefined (note that a vendor may choose to provide more information about the meaning of these pixels via control mechanisms, described later). See the description for the RENDERMODEL type and the `RenderVideo` request.

Windows created with `VideoGray` display only the luminance portion of a video signal; windows created with `VideoColor` are able to display a color picture if chrominance information is available in the video signal. A side effect of this visual class is that the visible rendition of regions not covered by a `RenderVideo` request are undefined, including the window’s border. However, it is expected that server implementors will pick reasonable defaults.

4. Types

VIDEOIN: 32-bit value (top three bits guaranteed to be zero)

VIDEOOUT: 32-bit value (top three bits guaranteed to be zero)

VIDEOIO: VIDEOIN or VIDEOUT

When a VIDEOIN or VIDEOOUT is supplied to a request, it must be one returned by the `CreateVideo` request, else `Match` error.

VREFERENCE: 32-bit value (top three bits guaranteed to be zero)

This is a special id supplied in a VIDEOGEOMETRY structure that uniquely identifies the corresponding resource.

FRACTION: [numerator, denominator: INT32]

The FRACTION type is used to represent a rational number; the denominator must be strictly positive (> 0) in all requests, else `Value` error.

FRACTIONRANGE: [num-base, num-inc, num-limit: INT32
num-type: {`Linear`, `Geometric`}
denom-base, denom-inc, denom-limit: INT32
denom-type: {`Linear`, `Geometric`}]

A FRACTIONRANGE is used in a PLACEMENT, described below. The set of fractions represented by FRACTIONRANGE depend on the types, and for each type, the range of numerators and denominators of each fraction are as follows:

`Linear`: n : {n = base to limit step inc

`Geometric`: n^{exp} : {n = base; exp = inc to limit step 1

A FRACTIONRANGE supplied by `QueryVideo` is guaranteed never to generate a fraction with a zero denominator, or a fractional numerator or denominator. For `Linear`, *num-inc* and *denom-inc* are guaranteed to be greater than zero, and the limit is always guaranteed to be -*base* plus some integral multiple of -*inc*. For `Geometric`, *num-inc* and *denom-inc* are guaranteed to be greater than or equal to zero. For example, a FRACTIONRANGE of {512, 128, 1024, `Linear`, 512, 512, 1024, `Linear`} creates a set containing

$$\left\{\frac{512}{512}, \frac{640}{512}, \frac{768}{512}, \frac{896}{512}, \frac{1024}{512}, \frac{512}{1024}, \frac{640}{1024}, \frac{768}{1024}, \frac{896}{1024}, \frac{1024}{1024}\right\}.$$

A FRACTIONRANGE of {2, 0, 2, `Geometric`, 1, 1, 127, `Linear`} creates the set

$$\left\{\left(\frac{2^0}{1} = \right)\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{127}, \frac{2}{1}, \frac{2}{2}, \ldots, \frac{2}{127}, \frac{4}{1}, \frac{4}{2}, \ldots, \frac{4}{127}\right\}.$$

MARKER: [kind: {Timecode, Frame, None}
relative: BOOL
data: TIMECODE or FRAME or empty]

The MARKER is used by `RenderVideo`, `CaptureGraphics` and `StopVideo` to indicate a relative or absolute point when the request will take effect. If Timecode is specified, then the data is a TIMECODE structure. If Frame is specified, then the data is a FRAME structure. If None is specified, then the data is empty. If relative is true, then the marker is relative to the current timecode or frame; otherwise it designates an absolute timecode or frame. If relative is false, then the negative element in the TIMECODE or FRAME must be false.

Whether timecodes or frame numbers are available from the video media is outside the domain of MVEX; i.e., another system must be used to obtain media information and events.

FRAME: [negative: BOOL
frame: CARD32
field: CARD8]

The FRAME is used by the MARKER type, and indirectly by `RenderVideo`, `CaptureGraphics` and `StopVideo` to indicate a relative or absolute frame when the request will take effect. If negative is true, then it refers to a negative frame number.

TIMECODE: [negative: BOOL
hour, minute, second, frame, field: CARD8]

The TIMECODE is used by the MARKER type, and indirectly by `RenderVideo`, `CaptureGraphics` and `StopVideo` to indicate a relative or absolute time point when the request will take effect. If negative is true, then it refers to a negative timecode.

RECTANGLERANGE: [base, limit: RECTANGLE
x-inc, y-inc: INT16
width-inc, height-inc: CARD16
type: {`Linear`, `Geometric`}]

The RECTANGLERANGE is ultimately used in the VIDEOGEOMETRY type below. It describes the possible geometries of sources and destinations for `RenderVideo` and `CaptureGraphics` requests. The set of rectangles described by a single RECTANGLERANGE are all combinations of four sets derived for x, y, width and height. The members of the sets are determined based on the type as follows.

`Linear`: x = { base.x, base.x + x-inc, base.x + 2·x-inc, ..., limit.x }
y = { base.y, base.y + y-inc, base.y + 2·y-inc, ..., limit.y }
width = { base.width, base.width + width-inc, base.width + 2·width-inc, ..., limit.width }
height = { base.height, base.height + height-inc, base.height + 2·height-inc, ..., limit.height }

`Geometric`: x = { $\text{base.x}^{\text{x-inc}}, \text{base.x}^{\text{x-inc+1}}, \ldots, \text{base.x}^{\text{limit.x}}$ }
y = { $\text{base.y}^{\text{y-inc}}, \text{base.y}^{\text{y-inc+1}}, \ldots, \text{base.y}^{\text{limit.y}}$ }
width = { $\text{base.width}^{\text{width-inc}}, \text{base.width}^{\text{width-inc+1}}, \ldots, \text{base.width}^{\text{limit.width}}$ }
height = { $\text{base.height}^{\text{height-inc}}, \text{base.height}^{\text{height-inc+1}}, \ldots, \text{base.height}^{\text{limit.height}}$ }

A RECTANGLERANGE supplied by `QueryVideo` is guaranteed never to generate a rectangle with a fractional component. For `Linear`, the *width-inc* and *height-inc* values are guaranteed to be greater than zero, and the *x-inc* and *y-inc* are guaranteed to be non-zero. For example, the RECTANGLERANGE whose value is

[base = {0, 0, 320, 240}, limit = {960, 784, 320, 240}
x-inc = 16, y-inc = 1
width-inc = 0, height-inc = 0
type = `Linear`]

would describe a total of 47,885 {x, y, width, height} sets

{0, 0, 320, 240}
{16, 0, 320, 240}
...
{960, 0, 320, 240}
{0, 1, 320, 240}
{16, 1, 320, 240}
...
{960, 1, 320, 240}
etc.
{960, 784, 320, 240}

PLACEMENT: [frame-rate: FRACTION
source, destination: RECTANGLERANGE
x-scale, y-scale: FRACTIONRANGE
identity-aspect: BOOL]

The PLACEMENT type is used in the VIDEOGEOMETRY type described below. Each placement describes the possible source and destination rectangles used in `RenderVideo` and `CaptureGraphics` requests, but restricted to the scale factors described by the x-scale and y-scale elements. For video input, the frame-rate element describes the maximum nominal frame rate that the hardware can achieve in a `RenderVideo` request using a source and destination rectangle derived from the other elements. For video output, the frame-rate element describes the maximum nominal rate at which frames can be encoded with a `CaptureGraphics` request. It is important to stress that this is only a nominal rate and may vary due to other factors such as hardware contention.

The x-scale and y-scale FRACTIONRANGE specifies the possible ratios of
destination x pixels : source x pixels
destination y pixels : source y pixels
where the ratio is numerator (destination) divided by denominator (source). The source and destination rectangles derived must have width and height ratios described by x-scale and y-scale. The identity-aspect is true if the ratio of x-scale to y-scale is 1:1; i.e. they must be equal. For example, given a PLACEMENT of

frame-rate = [30, 1]
source = [base = {0, 0, 10, 15}, limit = {639, 479, 640, 480}
x-inc = 1, y-inc = 1
width-inc = 1, height-inc = 1
type = `Linear`]
destination = [base = {0, 0, 320, 240}, limit = {960, 784, 320, 240}
x-inc = 16, y-inc = 1
width-inc = 0, height-inc = 0
type = `Linear`]
x-scale = [num-{base,inc,limit} = {2, 0, 6}
type = `Geometric`
denom-{base,inc,limit} = {1, 1, 0}
type = `Linear`]
y-scale = [num-{base,inc,limit} = {2, 0, 5}
type = `Geometric`
denom-{base,inc,limit} = {1, 1, 0}
type = `Linear`]
identity-aspect = False

it implies that the source rectangles may be any with the constraints that

$$0 \le x \le 639$$

$$0 \le y \le 479$$

$$10 \le \text{width} \le 640$$

$$15 \le \text{height} \le 480$$

and the destination rectangles may be any selected from the earlier example. The combined rectangles are limited to those source-destination pairs where the fractions

$$x-scale = \frac{\text{destination width}}{\text{source width}} \quad \text{and} \quad y-scale = \frac{\text{destination height}}{\text{source height}}$$

can be reduced to one of the fractions from the respective sets

$$x\text{-}scale = \left\{ \frac{1}{1}, \frac{2}{1}, \frac{4}{1}, \frac{8}{1}, \frac{16}{1}, \frac{32}{1}, \frac{64}{1} \right\}.$$

$$x\text{-}scale = \left\{ \frac{1}{1}, \frac{2}{1}, \frac{4}{1}, \frac{8}{1}, \frac{16}{1}, \frac{32}{1} \right\}.$$

And for this example, using any of these source-destination pairs, the hardware can maintain a nominal 30 fps.

VIDEOABILITY: [frames-available: BOOL
frame-accurate: BOOL
n-saturation: CARD32
n-contrast: CARD32
n-hue: CARD32
n-bright: CARD32
saturation: LISTofFRACTIONRANGE
contrast: LISTofFRACTIONRANGE
hue: LISTofFRACTIONRANGE
bright: LISTofFRACTIONRANGE]

This type describes the abilities of video inputs and video outputs and is returned by the QueryVideoRequest. The frames-available is true if the videoio is capable of recognizing frames or timecodes. The frame-accurate is true if `RenderVideo`, `CaptureGraphics` and `StopVideo` can be performed on precise frame boundaries. The n-saturation, n-contrast, n-hue and n-bright length of the corresponding FRACTIONRANGE structures. If a list has zero length, then adjustments and queries are ignored.

Saturation is the amount of color information present in the video image. Small values result in black and white images; large values provide increasing amounts of color.

Contrast determines the dynamic range of information present in the video image. Small values result in an image of constant color and intensity; large values produce more contrast.

Hue shifts the phase of the video’s color information relative to its reference, causing a shift in color. A middle value results in no shift, smaller values shift red towards blue and larger values shift red towards green.

Brightness determines the black level of the video signal. Small values result in a dimmer image, large values produce a brighter image.

VIDEOGEOMETRY: [signal-frame-rate: FRACTION
signal-field-rate: FRACTION
signal-width, signal-height: CARD16
concurrent-use: CARD16
reference-id: VREFERENCE
placement: LISTofPLACEMENT
ability: VIDEOABILITY]

This type describes the attributes of video inputs and video outputs and is returned by the `QueryVideo` request. The signal-frame-rate and signal-field-rate describe the number of frames per second and fields per second in the signal format. For input, the rate describes the signal received; for output, the signal produced.

For video input, the signal-width and signal-height describe the dimensions of the signal picture as if it were placed directly on the workstation screen. For video output, they describe the dimensions of a screen region whose pixels would map one-to-one to the output signal. It is important to note that the frame-buffer holding the video picture, if any, may not actually have these dimensions; the numbers describe the extent of the source rectangle for video input, or the destination rectangle for the video output.

The ability describes some specific abilities for a video input or output.

For a video input and output, the concurrent-use indicates the number of times the resource may be in simultaneous use, and suggests the maximum number identifiers a client should create using `CreateVideo`. The server should publish a number that the hardware can support with respect to connectivity and quality of the input or output. Clients should use this number in deciding how many resources they may use, and a video resource manager should use this in deciding how to handle allocation among multiple clients.

The reference id uniquely identifies the resource, and this id is used by the `CreateVideo` request to create video input and output identifiers. If the same reference id appears in a VIDEOGEOMETRY returned from different screens, then the concurrent use for will be the same and a video manager can infer that one physical resource services both screens. This implies that usage of this resource on one screen reduces the available usage on another screen.

Each placement element in the list describes its own range of values. The complete range of placement parameters is derived from the entire list, but parts of one element in the list may not be combined with another. For example, source and destination rectangles described in one placement may not be combined with scale factors described in another placement.

RENDERMODEL: [depth: CARD8
visual-id: VISUALID
opaque: BOOL
red-mask, green-mask, blue-mask: CARD32]

This type is used by `QueryVideo` to describe the depth and visual class of a potential source for `CaptureGraphics` or destination for `RenderVideo`. The visual id is one returned by the connection setup or the `QueryVideo` request. The boolean, *opaque*, is true when a video input is used to render to a drawable created with this visual id, and the pixel values created cannot be interpreted by a core visual class.

The red, green and blue mask are those used for `CaptureGraphics` when the source is a pixmap and the colormap is `None`; and for `RenderVideo` when the destination is a pixmap and *opaque* is false. A server may supply a RENDERMODEL with a visual id of None for depths that can only be captured from pixmaps; and a RENDERMODEL may be supplied with red-, green-, and blue-mask set to zero; but not both in the same RENDERMODEL.

It is conceivable that a system will support a depth for pixmaps, but not for windows. A server may support a gray scale `CaptureGraphics` for this depth by supplying a RENDERMODEL with visual id None and red- green- and blue- masks with identical bits set.

Note that if *opaque* is true, then the visual id is guaranteed to have a core visual class.

OWNER: [wid: WINDOW
vid: VIDEOIO]

This type is used by the `ChangeOwnership` request to establish ownership for any of the video resources: input, output and devices. The specified window identifies the client, and the videoid indicates the resource of which it gains ownership.

5. Errors

MVEX introduces one new error type.

Error	Description
`Video`	A value for a VIDEOIO argument does not name a VREFERENCE or provided by `QueryVideo`.

6. Requests

QueryVideo

wid: WINDOW
==>
screen: CARD8
video-depths: LISTofDEPTHS
allowed-depths: LISTofRENDERMODEL
inattr: LISTofVIDEOGEOMETRY
outattr: LISTofVIDEOGEOMETRY
video-input-models: LISTofBITMASK {`Pixmap`, `Window`}
video-output-models: LISTofBITMASK {`Pixmap`, `Window`, `Composite`