GoggleSprocket API v1.0
External ERS

December 10, 1996

Game Technology Group, Apple Computer
sprockets@adr.apple.com

Table Of Contents

Introduction

GoggleSprocket provides a device-independent interface to stereoscopic imaging devices. To display stereoscopic images with these drivers, you must use the DrawSprocket API v1.1 or later. You may use GoggleSprocket to allow your users to chose and configure their stereoscopic devices.

Data Types

```
typedef Boolean (*GSpEventProcPtr)( EventRecord *inEvent );
```

An event processing callback routine.

API Reference

OSStatus
GSpConfigure(
GSpEventProcPtr inEventProc,
Point *inUpperLeft);

Call this function to display a dialog that allows the user to choose and configure a stereoscopic imaging device.

OSStatus
GSpStartup(UInt32 inReserved);

Call this function before making any calls to GoggleSprocket.

OSStatus
GSpShutdown(UInt32 inReserved);

Call this function when you are finished using GoggleSprocket.

Stereoscopic Device Drivers

GoggleSprocket provides a device independent interface to stereoscopic devices through this device-driver API.

Driver Location and Creator Type

GoggleSprocket device drivers are shared libraries that reside in the Extensions folder, and have a creator type of ‘GSp ‘ (GSp followed by a space).

Required Resources

Every driver must have the following resurces defined or it will not be loaded.

CICN	128	Icon displayed in the configuration dialog within GSpConfigure().
STR	128	Name of the device supported. Displayed in the configuration dialog and used to identify the device that is active (this text is saved in a preferences file by GoggleSprocket).
VERS	128	Version of the driver. For drivers with the same name (‘STR ‘ 128), the driver with the latest version will be loaded.

Required Exports

The following exports must be present in your driver or it will not be loaded. They are defined in GoggleSprocketDevices.h.

```
OSStatus
GSpDevice_Open( void );
```

Opens the device driver in preparation for use. Perform any initialization required by the driver at this time.

```
OSStatus
GSpDevice_Close( void );
```

Closes the driver when it is no longer needed for use.

```
OSStatus
GSpDevice_Configure(
    Point     *inUpperLeft );
```

Present a configuration dialog. This function is called when the user chooses the “Setup” button from within the dialog presented by GSpConfigure().

```
OSStatus
GSpDevice_GetFirstModeInfo(
    GSpDeviceModeInfo       *outFirstModeInfo );
```

Return the first mode information structure supported by your device. Your device must

support at least one mode.

```
OSStatus
GSpDevice_GetNextModeInfo(
        GSpDeviceModeInfo       *inCurrentModeInfo,
        GSpDeviceModeInfo       *outNextModeInfo );
```

Return the next mode information structure that follows the input mode. This is the looping mechanism to iterate through the modes supported by your device.

```
OSStatus
GSpDevice_SetVisibleEye(
        Boolean         inLeftEyeVisible );
```

For devices that require manually setting the eye states, this function will tell your driver what eye to make visible (the eye that the user is seeing through). This function will only be called if the kGSpModeAttribute_ManualEyeToggleRequired bit is set in the mode attributes.

```
OSStatus
GSpDevice_GetVisibleEye(
        Boolean         *outLeftEyeVisible );
```

You must return the eye that the user is currently seeing through.

```
OSStatus
GSpDevice_CustomBufferProcess(
        CGrafPtr        inLeftEyeBuffer,
        CGrafPtr        inRightEyeBuffer,
        CGrafPtr        inDestBuffer,
        Boolean         inBuildLeftEyeBuffer,
        CTabHandle      ioColorTable,
        Boolean         *outShowDisplayBuffer,
        Boolean         *outCTabChanged );
```

This is the catch-all function that allows new device types to be added and supported by device drivers. If the kGSpModeAttribute_CustomProcess attribute flag is set for the mode, then this function will be called to do what is necessary to display the images.

This function will be called twice, first for the left eye and then the right eye. The inBuildLeftEye buffer indicates which eye is being processed.

Your device driver may process the image data, for example encoding sync information, or it may use the image data to display on it's own internal frame buffers.

If your driver is processing the images, but does not wish to display them itself, it should place the resulting image in the inDestBuffer and set the outShowDisplayBuffer flag to TRUE. You should place the processed image appropriate to the eye being processed in the output buffer.

If your device driver uses the left and right eye images to produce a single image that is stereoscopic (this is what the Apple Anaglyph Driver does), then you should perform the image processing when the driver is called to build the left eye, and return an error when requested to build the right eye.

If you change the color table, set the outCTabChanged flag to TRUE. This is not recommended, as most software using indexed graphics modes are very dependent on their color tables.

Constants

```
enum {
      kGSpDeviceKind_Unknown              = 0,
      kGSpDeviceKind_FrameSequential      = 1,
      kGSpDeviceKind_Stereoscopic         = 2
};
typedef UInt32 GSpDeviceKind;
```

kGSpDeviceKind_FrameSequential indicates a device that alternates eye frames, such as LCD shutter glasses.

kGSpDeviceKind_Stereoscopic indicates a device that can display the left and right eye frames simultaneously, such as a head mounted display.

```
enum {
      kGSpModeDataFormat_Unknown          = 0,
      kGSpModeDataFormat_Normal           = 1,
      kGSpModeDataFormat_Interleaved      = 2
};
typedef UInt32 GSpModeDataFormat;
```

kGSpModeDataFormat_Normal indicates that each eye frame occupies a full buffer.

kGSpModeDataFormat_Interleaved indicates that the eye frames are interleaved on a per-scanline basis.

```
enum {
      kGSpModeAttribute_CustomProcess           = (1L << 0L),
      kGSpModeAttribute_FullDisplayRequired     = (1L << 1L),
      kGSpModeAttribute_ManualEyeToggleRequired = (1L << 2L),
      kGSpModeAttribute_RightEyeDataFirst       = (1L << 3L),
      kGSpModeAttribute_PrimeBuffers            = (1L << 4L)
};
typedef UInt32 GSpModeAttributes;
```

kGSpModeAttribute_CustomProcess indicates that the device driver must perform custom buffer processing.

kGSpModeAttribute_FullDisplayRequired indicates that the stereo device must have the entire display to itself.

kGSpModeAttribute_ManualEyeToggleRequired indicates that the driver must be told which of

the user's eyes is visble.

kGSpModeAttribute_RightEyeDataFirst indicates that when data is interleaved (or in any form where the left and right eye data is combined), the right eye data should be first. The default is to display the left eye data first.

kGSpModeAttribute_PrimeBuffers indicates that the driver would like the destination buffer to be filled with the left or right image data (as appropriate) before the call to GSpDevice_CustomBufferProcess(). If your driver only encodes a small amount of information (as does the SimulEyes LCD glasses driver), it is more efficient to allow the system to copy the eye-frame than to do it within the driver.

Data Types

```
struct GSpDeviceModeInfo {
      GSpModeDataFormat       format;
      GSpModeAttributes       attributes;
      DisplayIDType           displayID;
      UInt32                  displayWidthRange[2];
      UInt32                  displayHeightRange[2];
      UInt32                  displayDepthRange[2];
      Fixed                   displayFrequencyRange[2];
};
typedef struct GSpDeviceModeInfo GSpDeviceModeInfo;
```

This data structure describes a mode supported by the device.

format and attributes are described above.

displayID is a Display Manager Display ID, and should be non-zero if your device only operates with a specific display in the system. A value of zero indicates that your device will work with any display.

The range parameters are inclusive boundaries describing the restrictions of the modes you can operate within. For example, if your device only operates within a mode that has a refresh rate between 90hz and 120hz, you would set displayFrequencyRange[0] to 90 (Fixed), and displayFrequencyRange [1] to 120 (Fixed). If you have no restrictions, set both of the range boundaries to zero (not just one).

