SIGIL API Design

SIGIL was designed to be as simple as possible, with little or no need for run-time error checking on behalf of the user. For this reason, and to ensure that some errors do not go unnoticed, certain SIGIL functions encountering any kind of error will terminate the program noisily with a descriptive error message.

Initialization and Window

`void slWindow(int width, int height, const char *title, int fullScreen)`

This function initializes SIGIL and will create a window with the desired width and height, titled with the given string. If `fullScreen` is non-zero, the window created will be a full screen window. This should be the first SIGIL function you call. This function terminates noisily if a window is already initialized.

`void slClose()`

This function de-initializes SIGIL and closes the SIGIL window that was previously opened with `slWindow()`. You should call this function when your program ends. SIGIL permits you to close the current window and open a new one afterwards if desired. This function terminates noisily if no window is currently open.

`int slShouldClose()`

This function returns non-zero if the user has attempted to close the SIGIL window that was opened with a call to `slWindow()`. This can be used to determine if your program's main loop should terminate. This function terminates noisily if no window is currently open.

Input

`int slGetKey(int key)`

This function returns non-zero when the given key is pressed. Alphabetic character (including the space bar) keys can be specified by providing the corresponding character value (such as 'W', 'A', etc.). Non-numpad numeric character keys can be specified by providing the corresponding character value (such as '0', '1', etc.). Numpad keys, non-printing keys (such as CTRL, SHIFT, escape, etc.) are specified using one of the values given in the list below.

```
SL_KEY_ESCAPE
SL_KEY_ENTER
SL_KEY_TAB
SL_KEY_BACKSPACE
SL_KEY_INSERT
SL_KEY_DELETE
SL_KEY_RIGHT
SL_KEY_LEFT
SL_KEY_DOWN
SL_KEY_UP
SL_KEY_PAGE_UP
```

SL_KEY_PAGE_DOWN
SL_KEY_HOME
SL_KEY_END
SL_KEY_CAPS_LOCK
SL_KEY_SCROLL_LOCK
SL_KEY_NUM_LOCK
SL_KEY_PRINT_SCREEN
SL_KEY_PAUSE
SL_KEY_F1
SL_KEY_F2
SL_KEY_F3
SL_KEY_F4
SL_KEY_F5
SL_KEY_F6
SL_KEY_F7
SL_KEY_F8
SL_KEY_F9
SL_KEY_F10
SL_KEY_F11
SL_KEY_F12
SL_KEY_F13
SL_KEY_F14
SL_KEY_F15
SL_KEY_F16
SL_KEY_F17
SL_KEY_F18
SL_KEY_F19
SL_KEY_F20
SL_KEY_F21
SL_KEY_F22
SL_KEY_F23
SL_KEY_F24
SL_KEY_F25
SL_KEY_KEYPAD_0
SL_KEY_KEYPAD_1
SL_KEY_KEYPAD_2
SL_KEY_KEYPAD_3
SL_KEY_KEYPAD_4
SL_KEY_KEYPAD_5
SL_KEY_KEYPAD_6
SL_KEY_KEYPAD_7
SL_KEY_KEYPAD_8
SL_KEY_KEYPAD_9
SL_KEY_KEYPAD_DECIMAL
SL_KEY_KEYPAD_DIVIDE
SL_KEY_KEYPAD_MULTIPLY

SL_KEY_KEYPAD_SUBTRACT
SL_KEY_KEYPAD_ADD
SL_KEY_KEYPAD_ENTER
SL_KEY_KEYPAD_EQUAL
SL_KEY_LEFT_SHIFT
SL_KEY_LEFT_CONTROL
SL_KEY_LEFT_ALT
SL_KEY_LEFT_SUPER
SL_KEY_RIGHT_SHIFT
SL_KEY_RIGHT_CONTROL
SL_KEY_RIGHT_ALT
SL_KEY_RIGHT_SUPER

`int slGetMouseButton(int button)`

This function returns non-zero when the given mouse button is pressed. Button values are specified using one of the values given in the list below.

SL_MOUSE_BUTTON_1
SL_MOUSE_BUTTON_2
SL_MOUSE_BUTTON_3
SL_MOUSE_BUTTON_4
SL_MOUSE_BUTTON_5
SL_MOUSE_BUTTON_6
SL_MOUSE_BUTTON_7
SL_MOUSE_BUTTON_LEFT (equivalent to SL_MOUSE_BUTTON_1)
SL_MOUSE_BUTTON_RIGHT (equivalent to SL_MOUSE_BUTTON_2)
SL_MOUSE_BUTTON_MIDDLE (equivalent to SL_MOUSE_BUTTON_3)

`int slGetMouseX()`

This function returns the x-position of the mouse cursor.

`int slGetMouseY()`

This function returns the y-position of the mouse cursor.

Timing

`double slGetDeltaTime()`

This function returns a *delta time* value that is calculated at the conclusion of every `slRender()` call. This delta time value represents the time in seconds that elapsed since the previous call to `slRender()`. Game object speeds or other time-dependent values should be multiplied by this value to ensure smooth animations. Before the first call to `slRender()`, this function returns 0.01666667, which is what `slGetDeltaTime()` would return if your program were running at 60 frames per second.

`double slGetTime()`

This function returns the current time value since the program started, in floating-point seconds.

Rendering

`void slRender()`

This function swaps the display buffers, causing all rendered objects to be displayed. It is also responsible for clearing the window to the desired color, as set by slSetBackColor(). Additionally, it performs input polling for functions such as slGetKey(), and so it should be called regularly. Ideally, you should call slRender() at the end of your main rendering loop.

Color and Blending

`slSetBackColor(double red, double green, double blue)`

This function sets the color of the window background. Each colour component (red, green, and blue) specified should be in the range [0.0, 1.0]. The initial background colour is (0.0, 0.0, 0.0). A call to slRender() is necessary to update the window background color.

`slSetForeColor(double red, double green, double blue, double alpha)`

This function sets the color of any objects that are drawn after this call is made. Each colour component (red, green, blue, and alpha transparency) specified should be in the range [0.0, 1.0]. The initial foreground colour is (1.0, 1.0, 1.0, 1.0).

`void slSetAdditiveBlend(int additiveBlend)`

This function enables or disables additive/intense blending for all objects that are drawn after this call is made, although blending itself is always enabled by SIGIL.

Transformations

`void slPush()`

This function pushes the current transformation matrix onto the matrix stack to allow for hierarchical transformations. This is useful for animation systems, moving a game camera, and other applications. It is analogous to the well-known (but deprecated) `glPush()` function provided by OpenGL.

`void slPop()`

This function pops the current transformation matrix off of the matrix stack to allow for hierarchical transformations. This is useful for animation systems, moving a game camera, and other applications. It is analogous to the well-known (but deprecated) `glPop()` function provided by OpenGL.

`void slTranslate(double x, double y)`

This function applies a translation matrix to the current matrix transformation. It is analogous to the well-known (but deprecated) `glTranslate()` functions provided by OpenGL. Transformations in SIGIL, as in OpenGL, are applied in the reverse order they are specified.

`void slRotate(double degrees)`

This function applies a rotation matrix to the current matrix transformation. It is analogous to the well-known (but deprecated) `glRotate()` functions provided by OpenGL. Transformations in SIGIL, as in OpenGL, are applied in the reverse order they are specified.

`void slScale(double x, double y)`

This function applies a scale matrix to the current matrix transformation. It is analogous to the well-known (but deprecated) `glScale()` functions provided by OpenGL. Transformations in SIGIL, as in OpenGL, are applied in the reverse order they are specified.

Texture Loading

`int slLoadTexture(const char *filename)`

This function loads the specified image file into texture memory and returns a unique integer identifier that can be passed to `slSprite()`. Supported file formats include BMP (non 1-bpp, non-RLE), PNG (non-interlaced), JPG (JPEG baseline), and TGA. This function terminates noisily if an unsupported format is detected or if the file could not be found.

Multiple calls to `slLoadTexture()` with the same filename are not optimized and will result in multiple copies of the same texture data with different integer identifiers. Therefore, it is recommended that you optimize your programs to only load each texture asset once and store the resulting integer identifiers in such a way that they can be accessed globally.

Sound Loading and Playing

`int slLoadWAV(const char *filename)`

This function loads the specified audio file and returns a unique integer identifier that can be passed to `slSoundPlay()` or `slSoundLoop()`. SIGIL only supports WAV file loading, and these files must be single channel and either 8 or 16 bits. This function terminates noisily if an unsupported format is detected or if the file could not be found.

Multiple calls to `slLoadWAV()` with the same filename are not optimized and will result in multiple copies of the same sound data with different integer identifiers. Therefore, it is recommended that you optimize your programs to only load each sound asset once and store the resulting integer identifiers in such a way that they can be accessed globally.

`int slSoundPlay(int sound)`

This function takes a sound integer identifier (that was returned by `slLoadWAV()`) and plays it once. It also returns a unique identifier that can be used as an argument to `slSoundPause()`, `slSoundStop()`, `slSoundPlaying()`, and `slSoundLooping()`.Identifiers returned by this function are re-used and are only valid until the sound finishes playing or up until `slSoundStop()` is called, whichever occurs first.

`int slSoundLoop(int sound)`

This function takes a unique sound integer identifier (that was returned by `slLoadWAV()`) and

loops it continuously. It also returns a unique identifier that can be used as an argument to `slSoundPause()`, `slSoundStop()`, `slSoundPlaying()`, and `slSoundLooping()`. Identifiers returned by this function are re-used and are only valid until `slSoundStop()` is called.

`void slSoundPause(int sound)`

This function takes a unique playing or looping sound identifier (that was returned by `slSoundPlay()` or `slSoundLoop()`) and pauses the sound associated with that identifier. The sound can be resumed by calling either `slSoundPlay()` or `slSoundLoop()` with the same identifier.

`void slSoundStop(int sound)`

This function takes a unique playing or looping sound identifier (that was returned by `slSoundPlay()` or `slSoundLoop()`) and stops the sound associated with that identifier. The identifier is also invalidated and freed for use by additional calls to `slSoundPlay()` or `slSoundLoop()`.

`void slSoundPauseAll()`

This function pauses all sounds that are currently playing or looping. Calling `slSoundResumeAll()` will resume any sounds that were paused either by `slSoundPauseAll()` or `slSoundPause()`.

`void slSoundStopAll()`

This function stops all sounds that are currently playing, looping, or paused, and invalidates any playing or looping sound identifiers returned by `slSoundPlay()` or `slSoundLoop()`.

`void slSoundResumeAll()`

This function resumes all sounds that were paused by `slSoundPauseAll()` or `slSoundPause()`.

`int slSoundPlaying(int sound)`

This function takes a unique playing or looping sound identifier (that was returned by `slSoundPlay()` or `slSoundLoop()`) and returns a non-zero value if and only if the identified sound is playing (but not looping) and not currently paused.

`int slSoundLooping(int sound)`

This function takes a unique playing or looping sound identifier (that was returned by `slSoundPlay()` or `slSoundLoop()`) and returns a non-zero value if and only if the identified sound is looping and not currently paused.

Shape Drawing

`void slTriangleFill(double x, double y, double width, double height)`

This function draws a filled triangle centered at the given coordinates, with the specified width and height. Transformation functions will affect how the object is rendered.

`void slTriangleOutline(double x, double y, double width, double height)`

This function draws a triangle outline centered at the given coordinates, with the specified width and height. Transformation functions will affect how the object is rendered.

`void slRectangleFill(double x, double y, double width, double height)`

This function draws a filled rectangle centered at the given coordinates, with the specified width and height. Transformation functions will affect how the object is rendered.

`void slRectangleOutline(double x, double y, double width, double height)`

This function draws a rectangle outline centered at the given coordinates, with the specified width and height. Transformation functions will affect how the object is rendered.

`void slCircleFill(double x, double y, double radius, int numVertices)`

This function draws a filled circle centered at the given coordinates, with the specified radius and vertex resolution. When drawing larger circles, you should use a larger number of vertices so the circle appears smooth. Transformation functions will affect how the object is rendered.

`void slCircleOutline(double x, double y, double radius, int numVertices)`

This function draws a circle outline centered at the given coordinates, with the specified radius and vertex resolution. When drawing larger circles, you should use a larger number of vertices so the circle appears smooth. Transformation functions will affect how the object is rendered.

`void slSemiCircleFill(double x, double y, double radius, int numVertices, double degrees)`

This function draws a filled semicircle centered at the given coordinates, with the specified radius and vertex resolution, across the specified number of degrees. When drawing larger semicircles, you should use a larger number of vertices so the circle appears smooth. Transformation functions will affect how the object is rendered. The value of `degrees` is clamped to [`-360.0`, `360.0`].

`void slSemiCircleOutline(double x, double y, double radius, int numVertices, double degrees)`

This function draws a semicircle outline centered at the given coordinates, with the specified radius and vertex resolution, across the specified number of degrees. When drawing larger semicircles, you should use a larger number of vertices so the circle appears smooth. Transformation functions will affect how the object is rendered. The value of `degrees` is clamped to [`-360.0`, `360.0`].

`void slPoint(double x, double y)`

This function draws a point at the given coordinates. Transformation functions will affect how the object is rendered.

`void slLine(double x1, double y1, double x2, double y2)`

This function draws a line between the points (x1, y1) and (x2, y2). Transformation functions will affect how the object is rendered.

`void slSetSpriteTiling(double x, double y)`

This function sets the amount that sprites rendered with `slSprite()` have their textures tiled. The default tiling value is (1.0, 1.0).

`void slSetSpriteScroll(double x, double y)`

This function sets the amount that sprites rendered with `slSprite()` have their textures offset. Normally, values are expected to range between 0.0 and 1.0, where 1.0 refers to the entire width/height of the texture. For example, calling `slSetSpriteScroll(0.5, 0.0)` will result in subsequent sprites having their textures offset halfway to the right.

`void slSprite(int texture, double x, double y, double width, double height)`

This functions draws an instance of a texture loaded by a previous call to `slLoadTexture()`, centered at the given coordinates, with the specified width and height. Transformation functions will affect how the object is rendered.

Text Drawing

`void slSetTextAlign(int textAlign)`

This function sets the text alignment for subsequent calls to `slText()`. Accepted values are `SL_ALIGN_LEFT`, `SL_ALIGN_CENTER`, and `SL_ALIGN_RIGHT`. The default alignment is `SL_ALIGN_LEFT`.

`double slGetTextWidth(const char *text)`

This function returns the width of the given character string, using the font specified by a previous call to `slSetFont()` (or `slSetFontSize()`, if a font type was already set).

`double slGetTextHeight(const char *text)`

This function returns the height of the given character string, using the font specified by a previous call to `slSetFont()` (or `slSetFontSize()` if a font type was already set).

`int slLoadFont(const char *filename)`

This function loads the specified font file and returns a unique integer identifier that can be passed to `slSetFont()`. This function terminates noisily if an unsupported format is detected or if the file could not be found.

Multiple calls to `slLoadFont()` with the same filename are not optimized and will result in multiple copies of the same font data with different integer identifiers. Therefore, it is recommended that you optimize your programs to only load each font asset once and store the resulting integer identifiers in such a way that they can be accessed globally.

`void slSetFont(int font, int fontSize)`

This function takes a font identifier that was returned by `slLoadFont()` and makes it the active font, with the specified point size. This function should be called before making calls to `slSetFontSize()`, `slText()`, `slGetTextWidth()`, or `slGetTextHeight()`.

`void slSetFontSize(int fontSize)`

This function sets the point size of the active font, which should have been specified by a previous call to `slSetFont()`. This function terminates noisily if no font has been set,

`void slText(double x, double y, const char *text)`

This function renders the given character string at the specified location, with the alignment specified by a previous call to `slSetTextAlign()` (or left-aligned by default).