**Initialization and Window**

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

This initializes SIGIL and will create a window of desired width and height, titled with the given string. This should be the very first SIGIL method you call.

**void slClose()**

This de-initializes SIGIL and closes the active window. You should call it when your program ends.

**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(). An ideal use case for this function is the following:

slWindow(...)

while(!slShouldClose())

{

// main loop code goes here

}

slClose();

**Input**

**int slGetKey(int key)**

This function returns non-zero when the given key is pressed. Alphabetic character 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 **key code table**.

**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 **mouse button table**.

**void slGetMousePos(int \*posX, int \*posY)**

This function takes two pointers to integer variables and fills them with the x and y position coordinates of the mouse cursor, regardless of whether or not the mouse in inside the active window. The returned mouse positions are clamped to the boundaries of the current monitor display, not the SIGIL window.

**Timing**

**float 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 (1/60 frames per second).

**Rendering**

**void slRender()**

This function swaps the display buffers, causing all rendered objects to be displayed. It should be called at the end of your main rendering loop.

**Color and Blending**

**slSetBackColor(float red, float green, float 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)

**slSetForeColor(float red, float green, float blue, float 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(float x, float 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(float 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(float x, float 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)

**Sound Loading and Playing**

**int slLoadWAV(const char \*filename)**

This function loads the specified audio file and returns a unique integer identifier. SIGIL only supports WAV file loading, and these files must be single channel and either 8 or 16 bits.

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 identifierr.

**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 or looping 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 or 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(float x, float y, float width, float height)

void slTriangleOutline(float x, float y, float width, float height)

void slRectangleFill(float x, float y, float width, float height)

void slRectangleOutline(float x, float y, float width, float height)

void slCircleFill(float x, float y, float radius, int numVertices)

void slCircleOutline(float x, float y, float radius, int numVertices)

void slPoint(float x, float y)

void slLine(float x1, float y1, float x2, float y2)

void slSetSpriteTiling(float x, float y)

void slSetSpriteScroll(float x, float y)

void slSprite(int texture, float x, float y, float width, float height)

**Text Drawing**

void slSetTextAlign(int textAlign)

float slGetTextWidth(const char \*text)

float slGetTextHeight(const char \*text)

void slSetFont(const char \*filename, int fontSize)

void slSetFontSize(int fontSize)

void slText(float x, float y, const char \*text)