**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 errors do not go unnoticed, SIGIL functions encountering any kind of error (including actions that would cause undefined behaviour) will terminate the program gracefully with a descriptive error message.

**Initialization and Window**

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

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

**void slClose()**

This function de-initializes SIGIL and closes the SIGIL window previously opened with slWindow(). You should call this function 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(). This can be used to determine if your program's main loop should terminate.

**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)

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.

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 indentifier in such a way that it 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.

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 it 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)**

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(float x, float y, float width, float 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(float x, float y, float width, float 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(float x, float y, float width, float 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(float x, float y, float 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(float x, float y, float 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 slPoint(float x, float y)**

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

**void slLine(float x1, float y1, float x2, float 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(float x, float 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(float x, float 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, float x, float y, float width, float 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\_TEXT\_ALIGN\_LEFT, SL\_TEXT\_ALIGN\_CENTER, and SL\_TEXT\_ALIGN\_RIGHT. The default alignment is SL\_TEXT\_ALIGN\_LEFT.

**float 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).

**float 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).

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

This function sets the active font from the given font TTF file, with the specified point size.

**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().

**void slText(float x, float 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).