SASL 2.4 change log – 21 Jun 2016

General notes

- Added multi-context sound support, can handle potentially infinite number of loaded samples.
- Added the ability to load SASL in a disabled state. This is done by placing a file *notselfenable.dat* in sasl/data/. Note that for SASL to work now you need to enable it externally.
- Algorithms for sound engine improved.
- Interpolation functions implementation changed.
- Added better OpenAL errors handling.
- Added key copy-paste ability for Mac (and optionally for Linux, requires xclip).
- Fixes for event intercepting window handling.
- Fixed onMouseMove event behavior for panel after click event.

X-Plane Scenery

float u, float v, float w = ModelToLocal(float x, float y, float z)

Converts model coordinates (with 0,0,0 at the center of the aircraft) into local openGL coordinates.

float x, float y, float z = LocalToModel(float u, float v, float w)

Converts local openGL coordinates into aircraft coordinates

draw3DLine(float x1, float y1, float z1, float x2, float y2, float z2)

Draws a 3D line between *x1,y1,z1* and *x2,y2,z2* – local coordinates

draw3DLine(float x1, float y1, float z1, float x2, float y2, float z2, float r, float g, float b, float a)

Draws a 3D line between *x1,y1,z1* and *x2,y2,z2* – local coordinates of color *r,g,b* and alpha *a*

draw3DCircle(float x, float y, float z, float r, int is_filled)

Draws a circle in *x,y,z* of radius *r*. *is_filled* indicates filled or not (0 or 1) orienteered to the camera.

draw3DCircle(float x, float y, float z, float r, int is_filled, float r, float g, float b, float a)

Draws a circle in *x,y,z* of radius *r*. *is_filled* indicates filled or not (0 or 1) of color *r,g,b* and alpha *a* orienteered to the camera.

draw3DCircle(float x, float y, float z, float r, int is_filled, float r, float g, float b, float a, float pitch, float yaw)

Draws a circle in *x,y,z* of radius *r*. *is_filled* indicates filled or not (0 or 1) of color *r,g,b,a* with *pitch* and *yaw*.

draw3DAngle(float x, float y, float z, float angle, float len, int rays)

Draws a 3D angle centered at *x,y,z*, of angular width *angle*, length *len* made out of *rays* rays . white and oriented with the nose of the model.

draw3DAngle(float x, float y, float z, float angle, float len, int rays, float r, float g, float b, float a)

Draws a 3D angle centered at *x,y,z*, of angular width *angle*, length *len* made out of *rays* rays , of color *r,g,b,a.* Oriented with the nose of the model.

draw3DAngle(float x, float y, float z, float angle, float len, int rays, float r, float g, float b, float a, float pitch, float yaw)

Draws a 3D angle centered at *x,y,z*, of angular width *angle*, length *len* made out of *rays* rays , of color *r,g,b,a.* with yaw *yaw* and pitch *pitch.*

draw3DStandingCone(float x, float y, float z, float r, float h)

Draws a standing up right white cone at *x,y,z* with radius *r* of height *h.*

draw3DStandingCone(float x, float y, float z, float r, float h, float r, float g, float b, float a)

Draws a standing up right cone at *x,y,z* with radius *r* of height *h*, of color *r,g,b,a.*

Component properties

Note: The following properties may be set inside the component by **set(param_name, true)** or by param_name = true in the component definition. Valid for both panel and pop-ups.

Note: any FBO assigned component will be automatically clipped by the edges of the component, i.e. simple clipping will be applied to the parameter *position* of the component. Use with caution, FBO creation and rendering requires resources, use *clip* property if you need simple clipping only.

Warning: Do not create FBOs in subcomponents of components with their own FBOs.

int fpslimit – FBO creating property

Sets the maximum number of calls of this component's draw() function. Note: setting this property initiates the creation of Framed Buffer Object which may impart performance, use with caution.

boolean noRenderSignal – FBO requiring property

Assign *true* in *update()* function of the component to skip rendering the next frame. If assigned, the next frame will simply redraw the render from the previous frame with no changes. Note: this constant requires FBO to have been created for the component beforehand. The assignment will last only one frame and will be automatically reset to *false.*

boolean mask – FBO creating property

Allows for masking functions to be used in *draw()* function of the component.

boolean clip

Sets simple clipping to the component, the component will be clipped by its *position* property.

[int, int, int, int] clip_size

Sets simple clipping to the required size. Requires *clip=true*. {lower left x,y , upper right x,y}

Sound

handle loadSample (string path)

Loads a *wav* sample from *path* and returns a handle to the sound.

handle loadSampleReversed(string path)

Loads a *wav* sample from *path* and returns a handle to the sound, the sound will be loaded in reverse.

handle loadSample(string path, int flag)

Loads a *wav* sample from *path* and returns a handle to the sound and creates and associated timer if *flag* is set to 1.

float getSamplePlayingRemaining (handle sampleID)

Returns how much time in seconds are left till the sound will stop *sample ID* playing if *sampleID* has an associated timer. Otherwise, returns 1 if the sound is playing and 0 if it is not.

Special Component Functions

Use the following construction inside draw callback of a component with FBO

drawMask()
-- Draw mask with primitives
drawUnderMask()
-- Draw under mask with primitives
drawMaskEnd()

Allows to draw a mask with primitives and then to draw under the mask. This construction can be called as many times as needed to create multiple level masks. Note: do not use this construction within itself.

setClipArea(int x1, int y1, int x2, int y2)

Sets the clip area of the component to the square defined by the two points. Do not use with clip_size

resetClipArea()

Disables clipping for the specified component

Use the following construct inside draw callback to get blending

setBlendEquation(const blendOp)
setBlendFunc(const srcBlend, const dstBlend)
-- Draw functions
resetBlending()
Allows to blend the source and destination using blendOP operation

Operations:
BLEND_EQUATION_MIN
BLEND_EQUATION_MAX
BLEND_EQUATION_SUBTRACT
BLEND_EQUATION_REVERSE_SUBTRACT
BLEND_EQUATION_ADD

Blending methods:
BLEND_SOURCE_COLOR
BLEND_ONE_MINUS_SOURCE_COLOR
BLEND_SOURCE_ALPHA
BLEND_ONE_MINUS_SOURCE_ALPHA
BLEND_DESTINATION_ALPHA
BLEND_ONE_MINUS_DESTINATION_ALPHA
BLEND_DESTINATION_COLOR
BLEND_ONE_MINUS_DESTINATION_COLOR
BLEND_SOURCE_ALPHA_SATURATE

Constant blend methods:
BLEND_CONSTANT_COLOR
BLEND_ONE_MINUS_CONSTANT_COLOR
BLEND_CONSTANT_ALPHA
BLEND_ONE_MINUS_CONSTANT_ALPHA

Constant methods require to set the color using **setBlendColor(float R, float G, float B, float A)**

The default is SASL is blendOP BLEND_EQUATION_ADD and BLEND_SOURCE_ALPHA, BLEND_ONE_MINUS_SOURCE_ALPHA

More info https://www.opengl.org/wiki/Blending

setBlendEquation(const blendOp)

Sets the blending operation to *blendOP*

setBlendFunc(const srcBlend, const dstBlend)

Sets the blending method to *scrBlend* for source and *dstBlend* for destination

setBlendFunc(const srcBlendRGB, const dstBlendRGB, const srcBlendAlpha, const dstBlendAlpha)

Sets the blending method analogous to the above but separately for RGB and alpha channels.

setBlendEquation(const blendOp)

Sets the blending method to *blendOP*

setBlendEquation(const blendOpRGB, const blendOpAlpha)

Sets the blending method to *blendOP* for RGB and *blendOpAlpha* for alpha

setBlendColor(float R, float G, float B, float A)

Sets the blend color for constant method blending

resetBlending()

Resets the blending to default

Graphics

drawCircle(float x, float y, float r)

Draws a circle in *x,y* of radius *r.*

drawCircle(float x, float y, float r, int seg)

Draws a circle in *x,y* of radius *r* using *seg* segments.

drawCircle(float x, float y, float r, int seg, float r, float g, float b)

Draws a circle in *x,y* of radius *r* of color *r,g,b* using *seg* segments.

drawCircle(float x, float y, float r, int seg, float r, float g, float b, float a)

Draws a circle in *x,y* of radius *r* of color *r,g,*b and alpha *a* using *seg* segments.

drawRotatedTextureCenter(handle id, float angle, float c_x, float c_y, int x, int y, int w, int h, float r, float g, float b, float a)

Draws the Texture *id* at *x,* y with width *w* and height *h* rotated by angle *angle* around the point *c_x, c_y.* The color is *r,g,b,a.*

drawTextureCoords(handle id, double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4, float r, float g, float b, float a)

Draws the Texture *id* at *coordinates x1, y1, x2, y2, x3, y3, x4, y4* . The color is *r,g,b,a.*

drawArc(double c_x, double c_y, double R1, double R2, double startAngle, double arcAngle)

Draws an arc with angle *arcAngle*, centered in *c_x, c_y* between radiuses *R1* and *R2* (*R1* < *R2*) starting at *startAngle.*

drawArc(double c_x, double c_y, double R1, double R2, double startAngle, double arcAngle, int seg)

Draws an arc with angle *arcAngle*, centered in *c_x, c_y* between radiuses *R1* and *R2* (*R1* < *R2*) starting at *startAngle* and using *seg* segments.

drawArc(double c_x, double c_y, double R1, double R2, double startAngle, double arcAngle, int seg, float r, float g, float b)

Draws an arc with angle *arcAngle*, centered in *c_x, c_y* between radiuses *R1* and *R2* (*R1 < R2*) starting at *startAngle* and using *seg* segments. The color is *r*, *g*, *b*.

drawArc(double c_x, double c_y, double R1, double R2, double startAngle, double arcAngle, int seg, float r, float g, float b, float a)

Draws an arc with angle *arcAngle*, centered in *c_x, c_y* between radiuses *R1* and *R2* (*R1 < R2*) starting at *startAngle* and using *seg* segments. The color is *r*, *g*, *b*, *a*.

Properties

xP = createGlobalPropertyi(string name, int default, int doNotPublish)

Create XP integer property *name* with value *default.* If *doNotPublish* is not provided or equals to 0 the name will be published.

xP = createGlobalPropertyf(string name, float default, int doNotPublish)

Create XP float property *name* with value *default.* If *doNotPublish* is not provided or equals to 0 the name will be published.

xP = createGlobalPropertys(string name, int maxLen, string default, int doNotPublish)

Create XP string property *name* with value *default.* If *doNotPublish* is not provided or equals to 0 the name will be published.

xP = createGlobalPropertyd(string name, double default, int doNotPublish)

Create XP double property *name* with value *default.* If *doNotPublish* is not provided or equals to 0 the name will be published.

xP = createGlobalSharedReferencei(string name, double default, int doNotPublish)

Create XP integer shared property *name* with value *default.* If *doNotPublish* is not provided or equals to 0 the name will be published.

xP = createGlobalSharedReferencef(string name, double default, int doNotPublish)

Create XP float shared property *name* with value *default.* If *doNotPublish* is not provided or equals to 0 the name will be published.

xP = createGlobalSharedReferences(string name, int maxLen, string default, int doNotPublish)

Create XP string shared property *name* with value *default.* If *doNotPublish* is not provided or equals to 0 the name will be published.

xP = createGlobalSharedReferenced(string name, double default, int doNotPublish)

Create XP double shared property *name* with value *default.* If *doNotPublish* is not provided or equals to 0 the name will be published.

All shared properties may be accessed for other plugins even after SASL is unloaded.

Utility

interp handle interp = newInterpolator(float array g1, float array g2…, data table T)

Creates a stepwise linear, interpolator from a grids *g1, g2,…,gn* which are n-dimensional vectors of variable lengths, and a result array of length *m* of *n*-dimensional matrices *T. T* is a "list" of result matrices representing a vector of results to interpolate given a point in n-dimensional space represented by the grids. Returns a handle to the interpolator.

float y = interpolate(float array x, interp handle interp)

Interpolates *x* using the interpolator *interp* returning the value *y.* Returns a number in case the interpolator had 1 value dimension and a vector otherwise. *x* can be passed as a number in case of one dimensional interpolation.

float y = interpolate(float array x, interp handle interp, int flag)

Interpolates *x* using the interpolator *interp* returning the value *y. flag* can be set to 0 to cut the interpolation at the edges or to 0 to extrapolate the value. Returns a number in case the interpolator had 1 value dimension and a vector otherwise. *x* can be passed as a number in case of one dimensional interpolation.

table t = table.merge(table t1, table t2)

Merges lua tables *t1* and *t2* into one table t, *t1* comes first.

string s = getAircraft()

Returns the full path, including the acf name to the loaded aircraft.

int getFrameCounter()

Returns XP frame number