IWEBGLRENDERER INTERFACE

View online: https://www.construct.net/en/make-games/manuals/addon-sdk/reference/graphics-interfaces/iwebglrenderer

The `IWebGLRenderer` interface provides methods for rendering to the Layout View, which is rendered using a canvas. The interface's methods provide high-level drawing commands implemented by Construct, so you don't need to handle low-level concerns like vertex buffers.

Despite the name, the renderer may in fact be backed by a WebGPU context. However this does not affect how the interface works.

This interface cannot be directly constructed. It is only available as a parameter to a `Draw()` call.

Renderer state

Since IWebGLRenderer is based on WebGL, it also uses a persistent rendering state. Therefore to correctly render something, all the intended state must be specified, otherwise it will use an undefined previous state. IWebGLRenderer simplifies the renderer state to:

1. A blend mode. Typically a normal alpha blend mode is used.
2. A fill mode (internally, the current fragment shader). The fill modes can be *color fill* (draw a solid color), *texture fill* (draw a texture), and *smooth line fill* (for drawing smooth lines).
3. A color set by `SetColor()` or `SetColorRgba()`. The alpha component of the color is used as the opacity in texture fill mode.
4. A texture set by `SetTexture()`. This is only used in texture fill mode.

Therefore a `Draw()` method should begin by specifying the blend mode, the fill mode, the color, and the texture (if texture fill mode is used), before continuing to draw. The renderer efficiently discards redundant calls, so if the state does not actually change then these calls have minimal performance overhead.

Once all state is set up, quads can be issued using one of the `Rect()` or `Quad()` method overloads. These methods draw using the currently set state.

Methods

SetAlphaBlendMode()

Set the blend mode to a premultiplied alpha blending mode.

SetBlendMode(blendMode)

Set the blend mode by a string which must be one of `"normal"`, `"additive"`, `"copy"`, `"destination-over"`, `"source-in"`, `"destination-in"`, `"source-out"`, `"destination-out"`, `"source-atop"`, `"destination-atop"`, `"lighten"`, `"darken"`, `"multiply"`, `"screen"`. Passing `"normal"` is equivalent to calling `SetAlphaBlendMode()`.

SetColorFillMode()

Set the fill mode to draw a solid color, specified by the current color.

SetTextureFillMode()

Set the fill mode to draw a texture, specified by the current texture, and using the alpha component of the current color as the opacity.

SetSmoothLineFillMode()

Set the fill mode to draw smooth lines using the current color.

SetColor(color)

Set the current color with an SDK.Color.

SetColorRgba(r, g, b, a)

Set the current color by directly passing the RGBA components.

SetOpacity(o)

Set only the alpha component of the current color. Note this does not affect the RGB components.

SetCurrentZ(z)

GetCurrentZ()

Set and get the current Z component used for all 2D drawing commands that don't specify Z components, such as the `Rect2()` and `Quad3()`.

ResetColor()

Set the current color to (1, 1, 1, 1).

Rect(rect)

Draw a rectangle given by an SDK.Rect.

Rect2(left, top, right, bottom)

Draw a rectangle by directly passing the left, top, right and bottom positions.

Quad(quad)

Draw a quad given by an SDK.Quad.

Quad2(tlx, tly, trx, try_, brx, bry, blx, bly)

Draw a quad by directly passing the positions of each of the four points in the quad.

Quad3(quad, rect)

Draw a quad given by an SDK.Quad, using an SDK.Rect for the source texture co-ordinates to draw from.

Quad4(quad, texQuad)

Draw a quad given by an SDK.Quad, using another `SDK.Quad` for the source texture co-ordinates to draw from.

Quad3D(tlx, tly, tlz, trx, try_, trz, brx, bry, brz, blx, bly, blz, rect)

Quad3D2(tlx, tly, tlz, trx, try_, trz, brx, bry, brz, blx, bly, blz, texQuad)

Draw a 3D quad, specifying all four points of the quad with X, Y and Z co-ordinates. The first overload accepts texture co-ordinates via an SDK.Rect *rect*, and the second accepts texture co-ordinates via an SDK.Quad *texQuad*.

DrawMesh(posArr, uvArr, indexArr, colorArr)

Draw an array of textured triangles based on the given position, texture co-ordinate and index arrays, and an optional per-vertex color array. For more details refer to the documentation for the IRenderer `drawMesh()` method, which works the same (albeit with different casing on the method name).

ConvexPoly(pointsArray)

Draw a convex polygon using the given array of points, in alternating X, Y order. Therefore the size of the array must be even, and must contain at least six elements (to define three points).

Line(x1, y1, x2, y2)

Draws a quad from the point (x1, y1) to (x2, y2) with the current line width.

TexturedLine(x1, y1, x2, y2, u, v)

Draws a quad from the point (x1, y1) to (x2, y2) with the current line width, and using (u, 0) as the texture co-ordinates at the start, and (v, 0) as the texture co-ordinates at the end.

LineRect(left, top, right, bottom)

Draws four lines along the edges of a given rectangle.

LineRect2(rect)

Draws four lines along the edges of a given `SDK.Rect`.

LineQuad(quad)

Draws four lines along the edges of a given `SDK.Quad`.

PushLineWidth(w)

PopLineWidth()

Set the current line width for line-drawing calls. This must be followed by a `PopLineWidth()` call when finished to restore the previous line width.

PushLineCap(lineCap)

PopLineCap()

Set the current line cap for line-drawing calls. This must be followed by a `PopLineCap()` call when finished to restore the previous line cap. The available line caps are `"butt"` and `"square"`.

SetTexture(texture)

Set the current texture to a given IWebGLTexture.

CreateWebGLText()

Return a new IWebGLText interface. This manages text wrapping, drawing text, and uploading the results to a WebGL texture.

CreateDynamicTexture(width, height, opts)

Create a new empty IWebGLTexture for dynamic use, i.e. expecting the texture content to be replaced using `UpdateTexture()`. The size of the texture is given by `width` and `height` which must be positive integers. `opts` specifies options for the texture which is an object that can include the following properties:

- `wrapX`: the texture horizontal wrap mode: one of `"clamp-to-edge"`, `"repeat"`, `"mirror-repeat"`
- `wrapY`: as with `wrapX` but for the vertical wrap mode
- `sampling`: the texture sampling mode, one of `"nearest"`, `"bilinear"` or `"trilinear"` (default)
- `pixelFormat`: the texture pixel format, one of `"rgba8"` (default), `"rgb8"`, `"rgba4"`, `"rgb5_a1"` or `"rgb565"`
- `mipMap`: boolean indicating if mipmaps should be used for this texture, default true
- `mipMapQuality`: if `mipMap` is true, one of `"default"` (default), `"low"` or `"high"`

UpdateTexture(data, texture, opts)

Upload *data* as the new texture contents for the IWebGLTexture *texture*. This can only be used for textures created with `CreateDynamicTexture()` and managed by your addon.
data can be one of the following types: `HTMLImageElement`, `HTMLVideoElement`, `HTMLCanvasElement`, `ImageBitmap`, `OffscreenCanvas` or `ImageData`. Note in worker mode the DOM types cannot be used (`HTMLImageElement`, `HTMLVideoElement`, `HTMLCanvasElement`); in this case use `ImageBitmap` or `OffscreenCanvas` instead. This method cannot resize an existing texture, so the data must match the size the texture was created with; if the size needs to change, destroy and re-create the texture.
opts specifies options for the texture upload which is an object that can include the following properties:

- `premultiplyAlpha`: a boolean indicating whether to premultiply alpha of the image content specified by *data* (default true). Construct always renders using premultiplied alpha so this is normally necessary; however if the data is known to already be premultiplied, set this to false.

DeleteTexture(texture)

Delete a IWebGLTexture, releasing its resources. This can only be used for textures created with `CreateDynamicTexture()` and managed by your addon. Do not attempt to delete textures managed by the Construct engine.