Skip to content

# GPU Types (gpu.types)

**class** gpu.types.**Buffer(format, dimensions, data)**

For Python access to GPU functions requiring a pointer.

**PARAMETERS:**

- **format** (*str*) – Format type to interpret the buffer. Possible values are *FLOAT*, *INT*, *UINT*, *UBYTE*, *UINT_24_8* and *10_11_11_REV*.
- **dimensions** (*int*) – Array describing the dimensions.
- **data** (*Buffer* | *Sequence[float]* | *Sequence[int]*) – Optional data array.

return the buffer as a list

**dimensions**

Undocumented, consider contributing.

**class** gpu.types.**GPUBatch(type, buf, elem=None)**

Reusable container for drawable geometry.

**PARAMETERS:**

- **type** (*str*) – The primitive type of geometry to be drawn. Possible values are *POINTS*, *LINES*, *TRIS*, *LINE_STRIP*, *LINE_LOOP*, *TRI_STRIP*, *TRI_FAN*, *LINES_ADJ*, *TRIS_ADJ* and *LINE_STRIP_ADJ*.
- **buf** (`gpu.types.GPUVertBuf`) – Vertex buffer containing all or some of the attributes required for drawing.
- **elem** (`gpu.types.GPUIndexBuf`) – An optional index buffer.

**draw(shader=None)**

Run the drawing shader with the parameters assigned to the batch.

**PARAMETERS:**

**shader** – Shader that performs the drawing operations. If `None` is passed, the last shader set to this batch will run.

**draw_instanced(program, *, instance_start=0, instance_count=0)**

Draw multiple instances of the drawing program with the parameters assigned to the batch. In the vertex shader, *gl_InstanceID* will contain tl
instance number being drawn.

**PARAMETERS:**

- **program** (`gpu.types.GPUShader`) – Program that performs the drawing operations.
- **instance_start** (*int*) – Number of the first instance to draw.
- **instance_count** (*int*) – Number of instances to draw. When not provided or set to 0 the number of instances will be determined by the number of rows in the first vertex buffer.

**draw_range(program, *, elem_start=0, elem_count=0)**

Run the drawing program with the parameters assigned to the batch. Only draw the `elem_count` elements of the index buffer starting at `elem_start`.

**PARAMETERS:**

- **program** (`gpu.types.GPUShader`) – Program that performs the drawing operations.
- **elem_start** (*int*) – First index to draw. When not provided or set to 0 drawing will start from the first element of the index buffer.
- **elem_count** (*int*) – Number of elements of the index buffer to draw. When not provided or set to 0 all elements from `elem_start` the end of the index buffer will be drawn.

**program_set(program)**

Assign a shader to this batch that will be used for drawing when not overwritten later. Note: This method has to be called in the draw context that the batch will be drawn in. This function does not need to be called when you always set the shader when calling `gpu.types.GPUBatch.draw()`.

**PARAMETERS:**

**program** ( `gpu.types.GPUShader` ) – The program/shader the batch will use in future draw calls.

**vertbuf_add(buf)**

Add another vertex buffer to the Batch. It is not possible to add more vertices to the batch using this method. Instead it can be used to add more attributes to the existing vertices. A good use case would be when you have a separate vertex buffer for vertex positions and vertex normals. Current a batch can have at most GPU_BATCH_VBO_MAX_LEN vertex buffers.

**PARAMETERS:**

**buf** ( `gpu.types.GPUVertBuf` ) – The vertex buffer that will be added to the batch.

**class** gpu.types.**GPUFrameBuffer(depth_slot=None, color_slots=None)**

This object gives access to framebuffer functionalities. When a 'layer' is specified in a argument, a single layer of a 3D or array texture is attached t
the frame-buffer. For cube map textures, layer is translated into a cube map face.

**PARAMETERS:**

- **depth_slot** ( `gpu.types.GPUTexture` | dict[] | None) – GPUTexture to attach or a *dict* containing keywords: 'texture', 'layer' and 'mip'.
- **color_slots** ( `gpu.types.GPUTexture` | dict[str, int | `gpu.types.GPUTexture` ] | Sequence[ `gpu.types.GPUTexture` | dict[str, int | `gpu.types.GPUTexture` ]] | None) – Tuple where each item can be a GPUTexture or a *dict* containing keywords: 'texture', 'layer' and 'mip'.

**bind()**

Context manager to ensure balanced bind calls, even in the case of an error.

**clear(color=None, depth=None, stencil=None)**

Fill color, depth and stencil textures with specific value. Common values: color=(0.0, 0.0, 0.0, 1.0), depth=1.0, stencil=0.

**PARAMETERS:**

- **color** (*Sequence[float]*) – Sequence of 3 or 4 floats representing `(r, g, b, a)`.
- **depth** (*float*) – depth value.
- **stencil** (*int*) – stencil value.

**read_color(x, y, xsize, ysize, channels, slot, format, data=data)**

Read a block of pixels from the frame buffer.

**PARAMETERS:**

- **y** (*x,*) – Lower left corner of a rectangular block of pixels.
- **ysize** (*xsize,*) – Dimensions of the pixel rectangle.
- **channels** (*int*) – Number of components to read.
- **slot** (*int*) – The framebuffer slot to read data from.
- **format** (*str*) – The format that describes the content of a single channel. Possible values are *FLOAT*, *INT*, *UINT*, *UBYTE*, *UINT_24_*
and *10_11_11_REV*.
- **data** ( `gpu.types.Buffer` ) – Optional Buffer object to fill with the pixels values.

**RETURNS:**

The Buffer with the read pixels.

**RETURN TYPE:**

`gpu.types.Buffer`

**read_depth(x, y, xsize, ysize, data=data)**

Read a pixel depth block from the frame buffer.

**PARAMETERS:**

- **y** (*x,*) – Lower left corner of a rectangular block of pixels.

- **ysize** (*xsize,*) – Dimensions of the pixel rectangle.
- **data** ( `gpu.types.Buffer` ) – Optional Buffer object to fill with the pixels values.

**RETURNS:**

The Buffer with the read pixels.

**RETURN TYPE:**

`gpu.types.Buffer`

**viewport_get()**

Returns position and dimension to current viewport.

**viewport_set(x, y, xsize, ysize)**

Set the viewport for this framebuffer object. Note: The viewport state is not saved upon framebuffer rebind.

**PARAMETERS:**

- **y** (*x,*) – lower left corner of the viewport_set rectangle, in pixels.
- **ysize** (*xsize,*) – width and height of the viewport_set.

**is_bound**

Checks if this is the active framebuffer in the context.

**class** gpu.types.**GPUIndexBuf(type, seq)**

Contains an index buffer.

**PARAMETERS:**

- **type** (*str*) – The primitive type this index buffer is composed of. Possible values are [ `POINTS` , `LINES` , `TRIS` , `LINES_ADJ` , `TRIS_ADJ` ].
- **seq** (*Buffer* | *Sequence[int]* | *Sequence[Sequence[int]]*) – Indices this index buffer will contain. Whether a 1D or 2D sequence is required depends on the type. Optionally the sequence can support the buffer protocol.

**class** gpu.types.**GPUOffScreen(width, height, *, format='RGBA8')**

This object gives access to off screen buffers.

**PARAMETERS:**

- **width** (*int*) – Horizontal dimension of the buffer.
- **height** (*int*) – Vertical dimension of the buffer.
- **format** (*str*) – Internal data format inside GPU memory for color attachment texture. Possible values are: *RGBA8*, *RGBA16*, *RGBA16F*, *RGBA32F*,

**bind()**

Context manager to ensure balanced bind calls, even in the case of an error.

**draw_view3d(scene, view_layer, view3d, region, view_matrix, projection_matrix, do_color_management=False, draw_background=True)**

Draw the 3d viewport in the offscreen object.

**PARAMETERS:**

- **scene** ( `bpy.types.Scene` ) – Scene to draw.
- **view_layer** ( `bpy.types.ViewLayer` ) – View layer to draw.
- **view3d** ( `bpy.types.SpaceView3D` ) – 3D View to get the drawing settings from.
- **region** ( `bpy.types.Region` ) – Region of the 3D View (required as temporary draw target).
- **view_matrix** ( `mathutils.Matrix` ) – View Matrix (e.g. `camera.matrix_world.inverted()` ).
- **projection_matrix** ( `mathutils.Matrix` ) – Projection Matrix (e.g. `camera.calc_matrix_camera(...)` ).
- **do_color_management** (*bool*) – Color manage the output.

- **draw_background** (*bool*) – Draw background.

**free()**

Free the offscreen object. The framebuffer, texture and render objects will no longer be accessible.

**unbind(restore=True)**

Unbind the offscreen object.

**PARAMETERS:**

**restore** (*bool*) – Restore the OpenGL state, can only be used when the state has been saved before.

**color_texture**

OpenGL bindcode for the color texture.

**TYPE:**

int

**height**

Height of the texture.

**TYPE:**

int

**texture_color**

The color texture attached.

**TYPE:**

`gpu.types.GPUTexture`

**width**

Width of the texture.

**TYPE:**

int

**class** gpu.types.**GPUShader(vertexcode, fragcode, geocode=None, libcode=None, defines=None, name='pyGPUShader')**

GPUShader combines multiple GLSL shaders into a program used for drawing. It must contain at least a vertex and fragment shaders.

The GLSL `#version` directive is automatically included at the top of shaders, and set to 330. Some preprocessor directives are automatically added according to the Operating System or availability: `GPU_ATI`, `GPU_NVIDIA` and `GPU_INTEL`.

The following extensions are enabled by default if supported by the GPU: `GL_ARB_texture_gather`, `GL_ARB_texture_cube_map_array` and `GL_ARB_shader_draw_parameters`.

For drawing user interface elements and gizmos, use `fragOutput = blender_srgb_to_framebuffer_space(fragOutput)` to transform the output sRGB colors to the frame-buffer color-space.

**PARAMETERS:**

- **vertexcode** (*str*) – Vertex shader code.
- **fragcode** – Fragment shader code.
- **geocode** – Geometry shader code.
- **libcode** – Code with functions and presets to be shared between shaders.
- **defines** – Preprocessor directives.
- **name** – Name of shader code, for debugging purposes.

**attr_from_name(name)**

Get attribute location by name.

**PARAMETERS:**

**name** (*str*) – The name of the attribute variable whose location is to be queried.

**RETURNS:**

The location of an attribute variable.

**RETURN TYPE:**

int

**attrs_info_get()**

Information about the attributes used in the Shader.

**RETURNS:**

tuples containing information about the attributes in order (name, type)

**RETURN TYPE:**

tuple[tuple[str, str | None], …]

**bind()**

Bind the shader object. Required to be able to change uniforms of this shader.

**format_calc()**

Build a new format based on the attributes of the shader.

**RETURNS:**

vertex attribute format for the shader

**RETURN TYPE:**

`gpu.types.GPUVertFormat`

**image(name, texture)**

Specify the value of an image variable for the current GPUShader.

**PARAMETERS:**

- **name** (*str*) – Name of the image variable to which the texture is to be bound.
- **texture** (`gpu.types.GPUTexture`) – Texture to attach.

**uniform_block(name, ubo)**

Specify the value of an uniform buffer object variable for the current GPUShader.

**PARAMETERS:**

- **name** (*str*) – name of the uniform variable whose UBO is to be specified.
- **ubo** – Uniform Buffer to attach.

**uniform_block_from_name(name)**

Get uniform block location by name.

**PARAMETERS:**

**name** (*str*) – Name of the uniform block variable whose location is to be queried.

**RETURNS:**

The location of the uniform block variable.

**RETURN TYPE:**

int

**uniform_bool(name, value)**

Specify the value of a uniform variable for the current program object.

**PARAMETERS:**

PARAMETERS:

- **name** (*str*) – Name of the uniform variable whose value is to be changed.
- **value** (*bool* | *Sequence[bool]*) – Value that will be used to update the specified uniform variable.

**uniform_float(name, value)**

Specify the value of a uniform variable for the current program object.

**PARAMETERS:**

- **name** (*str*) – Name of the uniform variable whose value is to be changed.
- **value** (*float* | *Sequence[float]*) – Value that will be used to update the specified uniform variable.

**uniform_from_name(name)**

Get uniform location by name.

**PARAMETERS:**

**name** (*str*) – Name of the uniform variable whose location is to be queried.

**RETURNS:**

Location of the uniform variable.

**RETURN TYPE:**

int

**uniform_int(name, seq)**

Specify the value of a uniform variable for the current program object.

**PARAMETERS:**

- **name** (*str*) – name of the uniform variable whose value is to be changed.
- **seq** (*Sequence[int]*) – Value that will be used to update the specified uniform variable.

**uniform_sampler(name, texture)**

Specify the value of a texture uniform variable for the current GPUShader.

**PARAMETERS:**

- **name** (*str*) – name of the uniform variable whose texture is to be specified.
- **texture** ( `gpu.types.GPUTexture` ) – Texture to attach.

**uniform_vector_float(location, buffer, length, count)**

Set the buffer to fill the uniform.

**PARAMETERS:**

- **location** (*int*) – Location of the uniform variable to be modified.
- **buffer** (*Sequence[float]*) – The data that should be set. Can support the buffer protocol.
- **length** (*int*) –

  Size of the uniform data type:

  - 1: float
  - 2: vec2 or float[2]
  - 3: vec3 or float[3]
  - 4: vec4 or float[4]
  - 9: mat3
  - 16: mat4
- **count** (*int*) – Specifies the number of elements, vector or matrices that are to be modified.

**uniform_vector_int(location, buffer, length, count)**

See GPUShader.uniform_vector_float(…) description.

**name**

The name of the shader object for debugging purposes (read-only).

**TYPE:**

str

**program**

The name of the program object for use by the OpenGL API (read-only).

**TYPE:**

int

**class** gpu.types.**GPUShaderCreateInfo**

Stores and describes types and variables that are used in shader sources.

**compute_source(source)**

compute shader source code written in GLSL.

Example:

```
"""void main() {
   int2 index = int2(gl_GlobalInvocationID.xy);
   vec4 color = vec4(0.0, 0.0, 0.0, 1.0);
   imageStore(img_output, index, color);
}"""
```

**PARAMETERS:**

**source** (*str*) – The compute shader source code.

See also

GLSL Cross Compilation

**define(name, value)**

Add a preprocessing define directive. In GLSL it would be something like:

```
#define name value

:arg name: Token name.
:type name: str
:arg value: Text that replaces token occurrences.
:type value: str
```

**fragment_out(slot, type, name, blend='NONE')**

Specify a fragment output corresponding to a framebuffer target slot.

**PARAMETERS:**

- **slot** (*int*) – The attribute index.
- **type** (*str*) –

  One of these types:

  - `FLOAT`
  - `VEC2`
  - `VEC3`
  - `VEC4`

    - `MAT3`
    - `MAT4`
    - `UINT`
    - `UVEC2`
    - `UVEC3`
    - `UVEC4`
    - `INT`
    - `IVEC2`
    - `IVEC3`
    - `IVEC4`
    - `BOOL`
- **name** (*str*) – Name of the attribute.
- **blend** (*str*) – Dual Source Blending Index. It can be 'NONE', 'SRC_0' or 'SRC_1'.

**fragment_source(source)**

Fragment shader source code written in GLSL.

Example:

```
"void main {fragColor = vec4(0.0, 0.0, 0.0, 1.0);}"
```

**PARAMETERS:**

**source** (*str*) – The fragment shader source code.

See also

GLSL Cross Compilation

**image(slot, format, type, name, qualifiers={'NO_RESTRICT'})**

Specify an image resource used for arbitrary load and store operations.

**PARAMETERS:**

- **slot** (*int*) – The image resource index.
- **format** (*str*) –

  The GPUTexture format that is passed to the shader. Possible values are:

    - `RGBA8UI`
    - `RGBA8I`
    - `RGBA8`
    - `RGBA32UI`
    - `RGBA32I`
    - `RGBA32F`
    - `RGBA16UI`
    - `RGBA16I`
    - `RGBA16F`
    - `RGBA16`
    - `RG8UI`
    - `RG8I`
    - `RG8`
    - `RG32UI`
    - `RG32I`
    - `RG32F`

    - `RG16UI`
    - `RG16I`
    - `RG16F`
    - `RG16`
    - `R8UI`
    - `R8I`
    - `R8`
    - `R32UI`
    - `R32I`
    - `R32F`
    - `R16UI`
    - `R16I`
    - `R16F`
    - `R16`
    - `R11F_G11F_B10F`
    - `DEPTH32F_STENCIL8`
    - `DEPTH24_STENCIL8`
    - `SRGB8_A8`
    - `RGB16F`
    - `SRGB8_A8_DXT1`
    - `SRGB8_A8_DXT3`
    - `SRGB8_A8_DXT5`
    - `RGBA8_DXT1`
    - `RGBA8_DXT3`
    - `RGBA8_DXT5`
    - `DEPTH_COMPONENT32F`
    - `DEPTH_COMPONENT24`
    - `DEPTH_COMPONENT16`
- **type** (*str*) –

  The data type describing how the image is to be read in the shader. Possible values are:

    - `FLOAT_BUFFER`
    - `FLOAT_1D`
    - `FLOAT_1D_ARRAY`
    - `FLOAT_2D`
    - `FLOAT_2D_ARRAY`
    - `FLOAT_3D`
    - `FLOAT_CUBE`
    - `FLOAT_CUBE_ARRAY`
    - `INT_BUFFER`
    - `INT_1D`
    - `INT_1D_ARRAY`
    - `INT_2D`
    - `INT_2D_ARRAY`
    - `INT_3D`
    - `INT_CUBE`
    - `INT_CUBE_ARRAY`
    - `UINT_BUFFER`
    - `UINT_1D`

- `UINT_1D_ARRAY`
- `UINT_2D`
- `UINT_2D_ARRAY`
- `UINT_3D`
- `UINT_CUBE`
- `UINT_CUBE_ARRAY`
- `SHADOW_2D`
- `SHADOW_2D_ARRAY`
- `SHADOW_CUBE`
- `SHADOW_CUBE_ARRAY`
- `DEPTH_2D`
- `DEPTH_2D_ARRAY`
- `DEPTH_CUBE`
- `DEPTH_CUBE_ARRAY`

- **name** (*str*) – The image resource name.
- **qualifiers** (*set[str]*) – Set containing values that describe how the image resource is to be read or written. Possible values are: - `NO_RESTRICT` - `READ` - `WRITE`

**local_group_size(x, y=-1, z=-1)**

Specify the local group size for compute shaders.

**PARAMETERS:**

- **x** (*int*) – The local group size in the x dimension.
- **y** (*int*) – The local group size in the y dimension. Optional. Defaults to -1.
- **z** (*int*) – The local group size in the z dimension. Optional. Defaults to -1.

**push_constant(type, name, size=0)**

Specify a global access constant.

**PARAMETERS:**

- **type** (*str*) –

  One of these types:

  - `FLOAT`
  - `VEC2`
  - `VEC3`
  - `VEC4`
  - `MAT3`
  - `MAT4`
  - `UINT`
  - `UVEC2`
  - `UVEC3`
  - `UVEC4`
  - `INT`
  - `IVEC2`
  - `IVEC3`
  - `IVEC4`
  - `BOOL`

- **name** (*str*) – Name of the constant.
- **size** (*int*) – If not zero, indicates that the constant is an array with the specified size.

**sampler(slot, type, name)**

Specify an image texture sampler.

**PARAMETERS:**

- **slot** (*int*) – The image texture sampler index.
- **type** (*str*) –

  The data type describing the format of each sampler unit. Possible values are:

  - `FLOAT_BUFFER`
  - `FLOAT_1D`
  - `FLOAT_1D_ARRAY`
  - `FLOAT_2D`
  - `FLOAT_2D_ARRAY`
  - `FLOAT_3D`
  - `FLOAT_CUBE`
  - `FLOAT_CUBE_ARRAY`
  - `INT_BUFFER`
  - `INT_1D`
  - `INT_1D_ARRAY`
  - `INT_2D`
  - `INT_2D_ARRAY`
  - `INT_3D`
  - `INT_CUBE`
  - `INT_CUBE_ARRAY`
  - `UINT_BUFFER`
  - `UINT_1D`
  - `UINT_1D_ARRAY`
  - `UINT_2D`
  - `UINT_2D_ARRAY`
  - `UINT_3D`
  - `UINT_CUBE`
  - `UINT_CUBE_ARRAY`
  - `SHADOW_2D`
  - `SHADOW_2D_ARRAY`
  - `SHADOW_CUBE`
  - `SHADOW_CUBE_ARRAY`
  - `DEPTH_2D`
  - `DEPTH_2D_ARRAY`
  - `DEPTH_CUBE`
  - `DEPTH_CUBE_ARRAY`
- **name** (*str*) – The image texture sampler name.

**typedef_source(source)**

Source code included before resource declaration. Useful for defining structs used by Uniform Buffers.

Example:

```
"struct MyType {int foo; float bar;};"

:arg source: The source code defining types.
:type source: str
```

**uniform_buf(slot, type_name, name)**

Specify a uniform variable whose type can be one of those declared in `gpu.types.GPUShaderCreateInfo.typedef_source()` .

**PARAMETERS:**

- **slot** (*int*) – The uniform variable index.
- **type_name** (*str*) – Name of the data type. It can be a struct type defined in the source passed through the `gpu.types.GPUShaderCreateInfo.typedef_source()` .
- **name** (*str*) – The uniform variable name.

**vertex_in(slot, type, name)**

Add a vertex shader input attribute.

**PARAMETERS:**

- **slot** (*int*) – The attribute index.
- **type** (*str*) –

  One of these types:

  - `FLOAT`
  - `VEC2`
  - `VEC3`
  - `VEC4`
  - `MAT3`
  - `MAT4`
  - `UINT`
  - `UVEC2`
  - `UVEC3`
  - `UVEC4`
  - `INT`
  - `IVEC2`
  - `IVEC3`
  - `IVEC4`
  - `BOOL`
- **name** (*str*) – name of the attribute.

**vertex_out(interface)**

Add a vertex shader output interface block.

**PARAMETERS:**

**interface** ( `gpu.types.GPUStageInterfaceInfo` ) – Object describing the block.

**vertex_source(source)**

Vertex shader source code written in GLSL.

Example:

```
"void main {gl_Position = vec4(pos, 1.0);}"
```

**PARAMETERS:**

**source** (*str*) – The vertex shader source code.

See also

GLSL Cross Compilation

**class** gpu.types.**GPUStageInterfaceInfo(name)**

List of varyings between shader stages.

**PARAMETERS:**

**name** – Name of the interface block.

**flat(type, name)**

Add an attribute with qualifier of type `flat` to the interface block.

**PARAMETERS:**

- **type** (*str*) –

  One of these types:

  - `FLOAT`
  - `VEC2`
  - `VEC3`
  - `VEC4`
  - `MAT3`
  - `MAT4`
  - `UINT`
  - `UVEC2`
  - `UVEC3`
  - `UVEC4`
  - `INT`
  - `IVEC2`
  - `IVEC3`
  - `IVEC4`
  - `BOOL`

- **name** (*str*) – name of the attribute.

**no_perspective(type, name)**

Add an attribute with qualifier of type `no_perspective` to the interface block.

**PARAMETERS:**

- **type** (*str*) –

  One of these types:

  - `FLOAT`
  - `VEC2`
  - `VEC3`
  - `VEC4`
  - `MAT3`
  - `MAT4`
  - `UINT`
  - `UVEC2`
  - `UVEC3`
  - `UVEC4`
  - `INT`
  - `IVEC2`
  - `IVEC3`

- `IVEC4`
- `BOOL`

- **name** (*str*) – name of the attribute.

**smooth(type, name)**

Add an attribute with qualifier of type *smooth* to the interface block.

**PARAMETERS:**

- **type** (*str*) –

  One of these types:

  - `FLOAT`
  - `VEC2`
  - `VEC3`
  - `VEC4`
  - `MAT3`
  - `MAT4`
  - `UINT`
  - `UVEC2`
  - `UVEC3`
  - `UVEC4`
  - `INT`
  - `IVEC2`
  - `IVEC3`
  - `IVEC4`
  - `BOOL`

- **name** (*str*) – name of the attribute.

**name**

Name of the interface block.

**TYPE:**

str

**class** gpu.types.**GPUTexture(size, layers=0, is_cubemap=False, format='RGBA8', data=None)**

This object gives access to off GPU textures.

**PARAMETERS:**

- **size** (*int | Sequence[int]*) – Dimensions of the texture 1D, 2D, 3D or cubemap.
- **layers** (*int*) – Number of layers in texture array or number of cubemaps in cubemap array
- **is_cubemap** (*int*) – Indicates the creation of a cubemap texture.
- **format** (*str*) – Internal data format inside GPU memory. Possible values are: *RGBA8UI*, *RGBA8I*, *RGBA8*, *RGBA32UI*, *RGBA32I*, *RGBA32F*, *RGBA16UI*, *RGBA16I*, *RGBA16F*, *RGBA16*, *RG8UI*, *RG8I*, *RG8*, *RG32UI*, *RG32I*, *RG32F*, *RG16UI*, *RG16I*, *RG16F*, *RG16*, *R8UI*, *R8I*, *R8*, *R32UI*, *R32I*, *R32F*, *R16UI*, *R16I*, *R16F*, *R16*, *R11F_G11F_B10F*, *DEPTH32F_STENCIL8*, *DEPTH24_STENCIL8*, *SRGB8_A8*, *RGB16F*, *SRGB8_A8_DXT1*, *SRGB8_A8_DXT3*, *SRGB8_A8_DXT5*, *RGBA8_DXT1*, *RGBA8_DXT3*, *RGBA8_DXT5*, *DEPTH_COMPONENT32F*, *DEPTH_COMPONENT24*, *DEPTH_COMPONENT16*,
- **data** (`gpu.types.Buffer`) – Buffer object to fill the texture.

**clear(format='FLOAT', value=(0.0, 0.0, 0.0, 1.0))**

Fill texture with specific value.

**PARAMETERS:**

- **format** (*str*) – The format that describes the content of a single item. Possible values are *FLOAT*, *INT*, *UINT*, *UBYTE*, *UINT_24_8* ar

*10_11_11_REV*.

- **value** (*Sequence[float]*) – Sequence each representing the value to fill. Sizes 1..4 are supported.

**read()**

Creates a buffer with the value of all pixels.

**format**

Format of the texture.

**TYPE:**

str

**height**

Height of the texture.

**TYPE:**

int

**width**

Width of the texture.

**TYPE:**

int

**class** gpu.types.**GPUUniformBuf(data)**

This object gives access to off uniform buffers.

**PARAMETERS:**

**data** (*object exposing buffer interface*) – Data to fill the buffer.

**update(data)**

Update the data of the uniform buffer object.

**class** gpu.types.**GPUVertBuf(format, len)**

Contains a VBO.

**PARAMETERS:**

- **format** (`gpu.types.GPUVertFormat`) – Vertex format.
- **len** (*int*) – Amount of vertices that will fit into this buffer.

**attr_fill(id, data)**

Insert data into the buffer for a single attribute.

**PARAMETERS:**

- **id** (*int* | *str*) – Either the name or the id of the attribute.
- **data** (*Buffer* | *Sequence[float]* | *Sequence[int]* | *Sequence[Sequence[float]]* | *Sequence[Sequence[int]]*) – Buffer or sequence of data that should be stored in the buffer

**class** gpu.types.**GPUVertFormat**

This object contains information about the structure of a vertex buffer.

**attr_add(id, comp_type, len, fetch_mode)**

Add a new attribute to the format.

**PARAMETERS:**

- **id** (*str*) – Name the attribute. Often *position*, *normal*, …
- **comp_type** (*str*) – The data type that will be used store the value in memory. Possible values are *I8*, *U8*, *I16*, *U16*, *I32*, *U32*, *F32* and

*I10.*

- **len** (*int*) – How many individual values the attribute consists of (e.g. 2 for uv coordinates).
- **fetch_mode** (*str*) – How values from memory will be converted when used in the shader. This is mainly useful for memory optimizations when you want to store values with reduced precision. E.g. you can store a float in only 1 byte but it will be converted to a normal 4 byt float when used. Possible values are *FLOAT*, *INT*, *INT_TO_FLOAT_UNIT* and *INT_TO_FLOAT*.