Skip to content

OpenGL Wrapper (bgl)

Warning

This module is deprecated and will be removed in a future release, when OpenGL is replaced by Metal and Vulkan. Use the graphics API independent `gpu` module instead.

This module wraps OpenGL constants and functions, making them available from within Blender Python.

The complete list can be retrieved from the module itself, by listing its contents: dir(bgl). A simple search on the web can point to more than enough material to teach OpenGL programming, from books to many collections of tutorials.

Here is a comprehensive list of books (non free). Learn OpenGL is one of the best resources to learn modern OpenGL and opengl-tutorial.org offers a s of extensive examples, including advanced features.

Note

You can use the `bpy.types.Image` type to load and set textures. See `bpy.types.Image.gl_load` and `bpy.types.Image.gl_free`, for example.

glBindTexture(target, texture):

Bind a named texture to a texturing target

See also

OpenGL Docs

PARAMETERS:

- **target** (*Enumerated constant*) – Specifies the target to which the texture is bound.
- **texture** (*unsigned int*) – Specifies the name of a texture.

glBlendFunc(sfactor, dfactor):

Specify pixel arithmetic

See also

OpenGL Docs

PARAMETERS:

- **sfactor** (*Enumerated constant*) – Specifies how the red, green, blue, and alpha source blending factors are computed.
- **dfactor** (*Enumerated constant*) – Specifies how the red, green, blue, and alpha destination blending factors are computed.

glClear(mask):

Clear buffers to preset values

See also

OpenGL Docs

PARAMETERS:

mask (*Enumerated constant(s)*) – Bitwise OR of masks that indicate the buffers to be cleared.

glClearColor(red, green, blue, alpha):

Specify clear values for the color buffers

See also

OpenGL Docs

PARAMETERS:

alpha (*red, green, blue,*) – Specify the red, green, blue, and alpha values used when the color buffers are cleared. The initial values are all 0.

glClearDepth(depth):

Specify the clear value for the depth buffer

See also

OpenGL Docs

PARAMETERS:

depth (*int*) – Specifies the depth value used when the depth buffer is cleared. The initial value is 1.

glClearStencil(s):

Specify the clear value for the stencil buffer

See also

OpenGL Docs

PARAMETERS:

s (*int*) – Specifies the index used when the stencil buffer is cleared. The initial value is 0.

glClipPlane (plane, equation):

Specify a plane against which all geometry is clipped

See also

OpenGL Docs

PARAMETERS:

- **plane** (*Enumerated constant*) – Specifies which clipping plane is being positioned.
- **equation** (`bgl.Buffer` object I{type GL_FLOAT}(double)) – Specifies the address of an array of four double- precision floating-point values. These values are interpreted as a plane equation.

glColorMask(red, green, blue, alpha):

Enable and disable writing of frame buffer color components

See also

OpenGL Docs

PARAMETERS:

alpha (*red, green, blue,*) – Specify whether red, green, blue, and alpha can or cannot be written into the frame buffer. The initial values are al GL_TRUE, indicating that the color components can be written.

glCopyTexImage2D(target, level, internalformat, x, y, width, height, border):

Copy pixels into a 2D texture image

See also

OpenGL Docs

PARAMETERS:

- **target** (*Enumerated constant*) – Specifies the target texture.
- **level** (*int*) – Specifies the level-of-detail number. Level 0 is the base image level. Level n is the nth mipmap reduction image.
- **internalformat** (*int*) – Specifies the number of color components in the texture.

- **y** (*x,*) – Specify the window coordinates of the first pixel that is copied from the frame buffer. This location is the lower left corner of a rectangular block of pixels.
- **width** (*int*) – Specifies the width of the texture image. Must be 2n+2(border) for some integer n. All implementations support texture images that are at least 64 texels wide.
- **height** (*int*) – Specifies the height of the texture image. Must be 2m+2(border) for some integer m. All implementations support texture image that are at least 64 texels high.
- **border** (*int*) – Specifies the width of the border. Must be either 0 or 1.

glCullFace(mode):

Specify whether front- or back-facing facets can be culled

See also

OpenGL Docs

PARAMETERS:

mode (*Enumerated constant*) – Specifies whether front- or back-facing facets are candidates for culling.

glDeleteTextures(n, textures):

Delete named textures

See also

OpenGL Docs

PARAMETERS:

- **n** (*int*) – Specifies the number of textures to be deleted
- **textures** (`bgl.Buffer` I{GL_INT}) – Specifies an array of textures to be deleted

glDepthFunc(func):

Specify the value used for depth buffer comparisons

See also

OpenGL Docs

PARAMETERS:

func (*Enumerated constant*) – Specifies the depth comparison function.

glDepthMask(flag):

Enable or disable writing into the depth buffer

See also

OpenGL Docs

PARAMETERS:

flag (*int (boolean)*) – Specifies whether the depth buffer is enabled for writing. If flag is GL_FALSE, depth buffer writing is disabled. Otherwise, it is enabled. Initially, depth buffer writing is enabled.

glDepthRange(zNear, zFar):

Specify mapping of depth values from normalized device coordinates to window coordinates

See also

OpenGL Docs

PARAMETERS:

- **zNear** (*int*) – Specifies the mapping of the near clipping plane to window coordinates. The initial value is 0.
- **zFar** (*int*) – Specifies the mapping of the far clipping plane to window coordinates. The initial value is 1.

glDisable(cap):

Disable server-side GL capabilities

See also

OpenGL Docs

PARAMETERS:

cap (*Enumerated constant*) – Specifies a symbolic constant indicating a GL capability.

glDrawBuffer(mode):

Specify which color buffers are to be drawn into

See also

OpenGL Docs

PARAMETERS:

mode (*Enumerated constant*) – Specifies up to four color buffers to be drawn into.

glEdgeFlag (flag):

B{glEdgeFlag, glEdgeFlagv}

Flag edges as either boundary or non-boundary

See also

OpenGL Docs

PARAMETERS:

flag (*Depends of function prototype*) – Specifies the current edge flag value.The initial value is GL_TRUE.

glEnable(cap):

Enable server-side GL capabilities

See also

OpenGL Docs

PARAMETERS:

cap (*Enumerated constant*) – Specifies a symbolic constant indicating a GL capability.

glEvalCoord (u,v):

B{glEvalCoord1d, glEvalCoord1f, glEvalCoord2d, glEvalCoord2f, glEvalCoord1dv, glEvalCoord1fv, glEvalCoord2dv, glEvalCoord2fv}

Evaluate enabled one- and two-dimensional maps

See also

OpenGL Docs

PARAMETERS:

- **u** (*Depends on function prototype.*) – Specifies a value that is the domain coordinate u to the basis function defined in a previous glMap1 or glMap2 command. If the function prototype ends in 'v' then u specifies a pointer to an array containing either one or two domain coordinates. The first coordinate is u. The second coordinate is v, which is present only in glEvalCoord2 versions.
- **v** (*Depends on function prototype. (only with '2' prototypes)*) – Specifies a value that is the domain coordinate v to the basis function defin in a previous glMap2 command. This argument is not present in a glEvalCoord1 command.

glEvalMesh (mode, i1, i2):

B{glEvalMesh1 or glEvalMesh2}

Compute a one- or two-dimensional grid of points or lines

See also

OpenGL Docs

PARAMETERS:

- **mode** (*Enumerated constant*) – In glEvalMesh1, specifies whether to compute a one-dimensional mesh of points or lines.
- **i2** (*i1,*) – Specify the first and last integer values for the grid domain variable i.

glEvalPoint (i, j):

B{glEvalPoint1 and glEvalPoint2}

Generate and evaluate a single point in a mesh

See also

OpenGL Docs

PARAMETERS:

- **i** (*int*) – Specifies the integer value for grid domain variable i.
- **j** (*int (only with '2' prototypes)*) – Specifies the integer value for grid domain variable j (glEvalPoint2 only).

glFeedbackBuffer (size, type, buffer):

Controls feedback mode

See also

OpenGL Docs

PARAMETERS:

- **size** (*int*) – Specifies the maximum number of values that can be written into buffer.
- **type** (*Enumerated constant*) – Specifies a symbolic constant that describes the information that will be returned for each vertex.
- **buffer** (`bgl.Buffer` object I{GL_FLOAT}) – Returns the feedback data.

glFinish():

Block until all GL execution is complete

See also

OpenGL Docs

glFlush():

Force Execution of GL commands in finite time

See also

OpenGL Docs

glFog (pname, param):

B{glFogf, glFogi, glFogfv, glFogiv}

Specify fog parameters

See also

OpenGL Docs

PARAMETERS:

- **pname** (*Enumerated constant*) – Specifies a single-valued fog parameter. If the function prototype ends in 'v' specifies a fog parameter.
- **param** (*Depends on function prototype.*) – Specifies the value or values to be assigned to pname. GL_FOG_COLOR requires an array of four values. All other parameters accept an array containing only a single value.

glFrontFace(mode):

Define front- and back-facing polygons

See also

OpenGL Docs

PARAMETERS:

mode (*Enumerated constant*) – Specifies the orientation of front-facing polygons.

glGenTextures(n, textures):

Generate texture names

See also

OpenGL Docs

PARAMETERS:

- **n** (*int*) – Specifies the number of textures name to be generated.
- **textures** (`bgl.Buffer` object I{type GL_INT}) – Specifies an array in which the generated textures names are stored.

glGet (pname, param):

B{glGetBooleanv, glGetfloatv, glGetFloatv, glGetIntegerv}

Return the value or values of a selected parameter

See also

OpenGL Docs

PARAMETERS:

- **pname** (*Enumerated constant*) – Specifies the parameter value to be returned.
- **param** (*Depends on function prototype.*) – Returns the value or values of the specified parameter.

glGetError():

Return error information

See also

OpenGL Docs

glGetLight (light, pname, params):

B{glGetLightfv and glGetLightiv}

Return light source parameter values

See also

OpenGL Docs

PARAMETERS:

- **light** (*Enumerated constant*) – Specifies a light source. The number of possible lights depends on the implementation, but at least eight lights are supported. They are identified by symbolic names of the form GL_LIGHTi where 0 < i < GL_MAX_LIGHTS.
- **pname** (*Enumerated constant*) – Specifies a light source parameter for light.
- **params** (`bgl.Buffer` object. Depends on function prototype.) – Returns the requested data.

glGetMap (target, query, v):

B{glGetMapdv, glGetMapfv, glGetMapiv}

Return evaluator parameters

See also

OpenGL Docs

PARAMETERS:

- **target** (*Enumerated constant*) – Specifies the symbolic name of a map.
- **query** (*Enumerated constant*) – Specifies which parameter to return.
- **v** (`bgl.Buffer` object. Depends on function prototype.) – Returns the requested data.

glGetMaterial (face, pname, params):

B{glGetMaterialfv, glGetMaterialiv}

Return material parameters

See also

OpenGL Docs

PARAMETERS:

- **face** (*Enumerated constant*) – Specifies which of the two materials is being queried. representing the front and back materials, respectively.
- **pname** (*Enumerated constant*) – Specifies the material parameter to return.
- **params** (`bgl.Buffer` object. Depends on function prototype.) – Returns the requested data.

glGetPixelMap (map, values):

B{glGetPixelMapfv, glGetPixelMapuiv, glGetPixelMapusv}

Return the specified pixel map

See also

OpenGL Docs

PARAMETERS:

- **map** (*Enumerated constant*) – Specifies the name of the pixel map to return.
- **values** (`bgl.Buffer` object. Depends on function prototype.) – Returns the pixel map contents.

glGetString(name):

Return a string describing the current GL connection

See also

OpenGL Docs

PARAMETERS:

- **name** (*Enumerated constant*) – Specifies a symbolic constant.

glGetTexEnv (target, pname, params):

B{glGetTexEnvfv, glGetTexEnviv}

Return texture environment parameters

See also

OpenGL Docs

PARAMETERS:

- **target** (*Enumerated constant*) – Specifies a texture environment. Must be GL_TEXTURE_ENV.
- **pname** (*Enumerated constant*) – Specifies the symbolic name of a texture environment parameter.
- **params** (`bgl.Buffer` object. Depends on function prototype.) – Returns the requested data.

glGetTexGen (coord, pname, params):

B{glGetTexGendv, glGetTexGenfv, glGetTexGeniv}

Return texture coordinate generation parameters

See also

OpenGL Docs

PARAMETERS:

- **coord** (*Enumerated constant*) – Specifies a texture coordinate.
- **pname** (*Enumerated constant*) – Specifies the symbolic name of the value(s) to be returned.
- **params** (`bgl.Buffer` object. Depends on function prototype.) – Returns the requested data.

glGetTexImage(target, level, format, type, pixels):

Return a texture image

See also

OpenGL Docs

PARAMETERS:

- **target** (*Enumerated constant*) – Specifies which texture is to be obtained.
- **level** (*int*) – Specifies the level-of-detail number of the desired image. Level 0 is the base image level. Level n is the nth mipmap reduction image.
- **format** (*Enumerated constant*) – Specifies a pixel format for the returned data.
- **type** (*Enumerated constant*) – Specifies a pixel type for the returned data.
- **pixels** (`bgl.Buffer` object.) – Returns the texture image. Should be a pointer to an array of the type specified by type

glGetTexLevelParameter (target, level, pname, params):

B{glGetTexLevelParameterfv, glGetTexLevelParameteriv}

return texture parameter values for a specific level of detail

See also

OpenGL Docs

PARAMETERS:

- **target** (*Enumerated constant*) – Specifies the symbolic name of the target texture.
- **level** (*int*) – Specifies the level-of-detail number of the desired image. Level 0 is the base image level. Level n is the nth mipmap reduction image.
- **pname** (*Enumerated constant*) – Specifies the symbolic name of a texture parameter.
- **params** (`bgl.Buffer` object. Depends on function prototype.) – Returns the requested data.

glGetTexParameter (target, pname, params):

B{glGetTexParameterfv, glGetTexParameteriv}

B{glGetTexParameterfv, glGetTexParameteriv}

Return texture parameter values

See also

OpenGL Docs

PARAMETERS:

- **target** (*Enumerated constant*) – Specifies the symbolic name of the target texture.
- **pname** (*Enumerated constant*) – Specifies the symbolic name the target texture.
- **params** (`bgl.Buffer` object. Depends on function prototype.) – Returns the texture parameters.

glHint(target, mode):

Specify implementation-specific hints

See also

OpenGL Docs

PARAMETERS:

- **target** (*Enumerated constant*) – Specifies a symbolic constant indicating the behavior to be controlled.
- **mode** (*Enumerated constant*) – Specifies a symbolic constant indicating the desired behavior.

glIsEnabled(cap):

Test whether a capability is enabled

See also

OpenGL Docs

PARAMETERS:

cap (*Enumerated constant*) – Specifies a constant representing a GL capability.

glIsTexture(texture):

Determine if a name corresponds to a texture

See also

OpenGL Docs

PARAMETERS:

texture (*unsigned int*) – Specifies a value that may be the name of a texture.

glLight (light, pname, param):

B{glLightf,glLighti, glLightfv, glLightiv}

Set the light source parameters

See also

OpenGL Docs

PARAMETERS:

- **light** (*Enumerated constant*) – Specifies a light. The number of lights depends on the implementation, but at least eight lights are supported. They are identified by symbolic names of the form GL_LIGHTi where 0 < i < GL_MAX_LIGHTS.
- **pname** (*Enumerated constant*) – Specifies a single-valued light source parameter for light.
- **param** (*Depends on function prototype.*) – Specifies the value that parameter pname of light source light will be set to. If function prototype ends in 'v' specifies a pointer to the value or values that parameter pname of light source light will be set to.

glLightModel (pname, param):

B{glLightModelf, glLightModeli, glLightModelfv, glLightModeliv}

Set the lighting model parameters

See also

OpenGL Docs

PARAMETERS:

- **pname** (*Enumerated constant*) – Specifies a single-value light model parameter.
- **param** (*Depends on function prototype.*) – Specifies the value that param will be set to. If function prototype ends in 'v' specifies a pointer t the value or values that param will be set to.

glLineWidth(width):

Specify the width of rasterized lines.

See also

OpenGL Docs

PARAMETERS:

width (*float*) – Specifies the width of rasterized lines. The initial value is 1.

glLoadMatrix (m):

B{glLoadMatrixd, glLoadMatixf}

Replace the current matrix with the specified matrix

See also

OpenGL Docs

PARAMETERS:

m (`bgl.Buffer` object. Depends on function prototype.) – Specifies a pointer to 16 consecutive values, which are used as the elements a 4x4 column-major matrix.

glLogicOp(opcode):

Specify a logical pixel operation for color index rendering

See also

OpenGL Docs

PARAMETERS:

opcode (*Enumerated constant*) – Specifies a symbolic constant that selects a logical operation.

glMap1 (target, u1, u2, stride, order, points):

B{glMap1d, glMap1f}

Define a one-dimensional evaluator

See also

OpenGL Docs

PARAMETERS:

- **target** (*Enumerated constant*) – Specifies the kind of values that are generated by the evaluator.
- **u1,u2** – Specify a linear mapping of u, as presented to glEvalCoord1, to ^, t he variable that is evaluated by the equations specified by this command

command.

- **stride** (*int*) – Specifies the number of floats or float (double)s between the beginning of one control point and the beginning of the next one in the data structure referenced in points. This allows control points to be embedded in arbitrary data structures. The only constraint is that the values for a particular control point must occupy contiguous memory locations.
- **order** (*int*) – Specifies the number of control points. Must be positive.
- **points** (`bgl.Buffer` object. Depends on function prototype.) – Specifies a pointer to the array of control points.

glMap2 (target, u1, u2, ustride, uorder, v1, v2, vstride, vorder, points):

B{glMap2d, glMap2f}

Define a two-dimensional evaluator

See also

OpenGL Docs

PARAMETERS:

- **target** (*Enumerated constant*) – Specifies the kind of values that are generated by the evaluator.
- **u1,u2** – Specify a linear mapping of u, as presented to glEvalCoord2, to ^, t he variable that is evaluated by the equations specified by this command. Initially u1 is 0 and u2 is 1.
- **ustride** (*int*) – Specifies the number of floats or float (double)s between the beginning of control point R and the beginning of control point R where i and j are the u and v control point indices, respectively. This allows control points to be embedded in arbitrary data structures. The or constraint is that the values for a particular control point must occupy contiguous memory locations. The initial value of ustride is 0.
- **uorder** (*int*) – Specifies the dimension of the control point array in the u axis. Must be positive. The initial value is 1.
- **v2** (*v1,*) – Specify a linear mapping of v, as presented to glEvalCoord2, to ^, one of the two variables that are evaluated by the equations specified by this command. Initially, v1 is 0 and v2 is 1.
- **vstride** (*int*) – Specifies the number of floats or float (double)s between the beginning of control point R and the beginning of control point R where i and j are the u and v control point(indices, respectively. This allows control points to be embedded in arbitrary data structures. The or constraint is that the values for a particular control point must occupy contiguous memory locations. The initial value of vstride is 0.
- **vorder** (*int*) – Specifies the dimension of the control point array in the v axis. Must be positive. The initial value is 1.
- **points** (`bgl.Buffer` object. Depends on function prototype.) – Specifies a pointer to the array of control points.

glMapGrid (un, u1,u2 ,vn, v1, v2):

B{glMapGrid1d, glMapGrid1f, glMapGrid2d, glMapGrid2f}

Define a one- or two-dimensional mesh

See also

OpenGL Docs

PARAMETERS:

- **un** (*int*) – Specifies the number of partitions in the grid range interval [u1, u2]. Must be positive.
- **u2** (*u1,*) – Specify the mappings for integer grid domain values i=0 and i=un.
- **vn** (*int*) – Specifies the number of partitions in the grid range interval [v1, v2] (glMapGrid2 only).
- **v2** (*v1,*) – Specify the mappings for integer grid domain values j=0 and j=vn (glMapGrid2 only).

glMaterial (face, pname, params):

Specify material parameters for the lighting model.

See also

OpenGL Docs

PARAMETERS:

- **face** (*Enumerated constant*) – Specifies which face or faces are being updated. Must be one of:
- **pname** (*Enumerated constant*) – Specifies the single-valued material parameter of the face or faces that is being updated. Must be

GL_SHININESS.

- **params** (*int*) – Specifies the value that parameter GL_SHININESS will be set to. If function prototype ends in 'v' specifies a pointer to the value or values that pname will be set to.

glMultMatrix (m):

B{glMultMatrixd, glMultMatrixf}

Multiply the current matrix with the specified matrix

See also

OpenGL Docs

PARAMETERS:

m (`bgl.Buffer` object. Depends on function prototype.) – Points to 16 consecutive values that are used as the elements of a 4x4 colum major matrix.

glNormal3 (nx, ny, nz, v):

B{Normal3b, Normal3bv, Normal3d, Normal3dv, Normal3f, Normal3fv, Normal3i, Normal3iv, Normal3s, Normal3sv}

Set the current normal vector

See also

OpenGL Docs

PARAMETERS:

- **nz** (*nx, ny,*) – Specify the x, y, and z coordinates of the new current normal. The initial value of the current normal is the unit vector, (0, 0, 1).
- **v** (`bgl.Buffer` object. Depends on function prototype. ('v' prototypes)) – Specifies a pointer to an array of three elements: the x, y, anc coordinates of the new current normal.

glPixelMap (map, mapsize, values):

B{glPixelMapfv, glPixelMapuiv, glPixelMapusv}

Set up pixel transfer maps

See also

OpenGL Docs

PARAMETERS:

- **map** (*Enumerated constant*) – Specifies a symbolic map name.
- **mapsize** (*int*) – Specifies the size of the map being defined.
- **values** (`bgl.Buffer` object. Depends on function prototype.) – Specifies an array of mapsize values.

glPixelStore (pname, param):

B{glPixelStoref, glPixelStorei}

Set pixel storage modes

See also

OpenGL Docs

PARAMETERS:

- **pname** (*Enumerated constant*) – Specifies the symbolic name of the parameter to be set. Six values affect the packing of pixel data into memory. Six more affect the unpacking of pixel data from memory.
- **param** (*Depends on function prototype.*) – Specifies the value that pname is set to.

glPixelTransfer (pname, param):

B{glPixelTransferf, glPixelTransferi}

Set pixel transfer modes

See also

OpenGL Docs

PARAMETERS:

- **pname** (*Enumerated constant*) – Specifies the symbolic name of the pixel transfer parameter to be set.
- **param** (*Depends on function prototype.*) – Specifies the value that pname is set to.

glPointSize(size):

Specify the diameter of rasterized points

See also

OpenGL Docs

PARAMETERS:

size (*float*) – Specifies the diameter of rasterized points. The initial value is 1.

glPolygonMode(face, mode):

Select a polygon rasterization mode

See also

OpenGL Docs

PARAMETERS:

- **face** (*Enumerated constant*) – Specifies the polygons that mode applies to. Must be GL_FRONT for front-facing polygons, GL_BACK for back- facing polygons, or GL_FRONT_AND_BACK for front- and back-facing polygons.
- **mode** (*Enumerated constant*) – Specifies how polygons will be rasterized. The initial value is GL_FILL for both front- and back- facing polygons.

glPolygonOffset(factor, units):

Set the scale and units used to calculate depth values

See also

OpenGL Docs

PARAMETERS:

- **factor** (*float*) – Specifies a scale factor that is used to create a variable depth offset for each polygon. The initial value is 0.
- **units** (*float*) – Is multiplied by an implementation-specific value to create a constant depth offset. The initial value is 0.

glRasterPos (x,y,z,w):

B{glRasterPos2d, glRasterPos2f, glRasterPos2i, glRasterPos2s, glRasterPos3d, glRasterPos3f, glRasterPos3i, glRasterPos3s, glRasterPos4d, glRasterPos4f, glRasterPos4i, glRasterPos4s, glRasterPos2dv, glRasterPos2fv, glRasterPos2iv, glRasterPos2sv, glRasterPos3dv, glRasterPos3fv, glRasterPos3iv, glRasterPos3sv, glRasterPos4dv, glRasterPos4fv, glRasterPos4iv, glRasterPos4sv}

Specify the raster position for pixel operations

See also

OpenGL Docs

PARAMETERS:

w(x, y, z) – Specify the x, y, z, and w object coordinates (if present) for the raster position. If function prototype ends in 'v' specifies a pointe

w (*x, y, z,*) – Specify the x,y,z, and w object coordinates (if present) for the raster position. If function prototype ends in 'v' specifies a pointer to an array of two, three, or four elements, specifying x, y, z, and w coordinates, respectively.

Note

If you are drawing to the 3d view with a Scriptlink of a space handler the zoom level of the panels will scale the glRasterPos by the view matrix. so a X of 10 will not always offset 10 pixels as you would expect.

To work around this get the scale value of the view matrix and use it to scale your pixel values.

```
import bgl
xval, yval= 100, 40
# Get the scale of the view matrix
view_matrix = bgl.Buffer(bgl.GL_FLOAT, 16)
bgl.glGetFloatv(bgl.GL_MODELVIEW_MATRIX, view_matrix)
f = 1.0 / view_matrix[0]

# Instead of the usual glRasterPos2i(xval, yval)
bgl.glRasterPos2f(xval * f, yval * f)
```

glReadBuffer(mode):

Select a color buffer source for pixels.

See also

OpenGL Docs

PARAMETERS:

mode (*Enumerated constant*) – Specifies a color buffer.

glReadPixels(x, y, width, height, format, type, pixels):

Read a block of pixels from the frame buffer

See also

OpenGL Docs

PARAMETERS:

- **y** (*x,*) – Specify the window coordinates of the first pixel that is read from the frame buffer. This location is the lower left corner of a rectangul block of pixels.
- **height** (*width,*) – Specify the dimensions of the pixel rectangle. width and height of one correspond to a single pixel.
- **format** (*Enumerated constant*) – Specifies the format of the pixel data.
- **type** (*Enumerated constant*) – Specifies the data type of the pixel data.
- **pixels** (`bgl.Buffer` object) – Returns the pixel data.

glRect (x1,y1,x2,y2,v1,v2):

B{glRectd, glRectf, glRecti, glRects, glRectdv, glRectfv, glRectiv, glRectsv}

Draw a rectangle

See also

OpenGL Docs

PARAMETERS:

- **y1** (*x1,*) – Specify one vertex of a rectangle
- **y2** (*x2,*) – Specify the opposite vertex of the rectangle
- **v2** (*v1,*) – Specifies a pointer to one vertex of a rectangle and the pointer to the opposite vertex of the rectangle

glRotate (angle, x, y, z):

B{glRotated, glRotatef}

Multiply the current matrix by a rotation matrix

See also

OpenGL Docs

PARAMETERS:

- **angle** (*Depends on function prototype.*) – Specifies the angle of rotation in degrees.
- **z** (*x, y,*) – Specify the x, y, and z coordinates of a vector respectively.

glScale (x,y,z):

B{glScaled, glScalef}

Multiply the current matrix by a general scaling matrix

See also

OpenGL Docs

PARAMETERS:

z (*x, y,*) – Specify scale factors along the x, y, and z axes, respectively.

glScissor(x,y,width,height):

Define the scissor box

See also

OpenGL Docs

PARAMETERS:

- **y** (*x,*) – Specify the lower left corner of the scissor box. Initially (0, 0).
- **height** (*width*) – Specify the width and height of the scissor box. When a GL context is first attached to a window, width and height are set tc the dimensions of that window.

glStencilFunc(func, ref, mask):

Set function and reference value for stencil testing

See also

OpenGL Docs

PARAMETERS:

- **func** (*Enumerated constant*) – Specifies the test function.
- **ref** (*int*) – Specifies the reference value for the stencil test. ref is clamped to the range [0,2n-1], where n is the number of bitplanes in the stenc buffer. The initial value is 0.
- **mask** (*unsigned int*) – Specifies a mask that is ANDed with both the reference value and the stored stencil value when the test is done. The initial value is all 1's.

glStencilMask(mask):

Control the writing of individual bits in the stencil planes

See also

OpenGL Docs

PARAMETERS:

mask (*unsigned int*) – Specifies a bit mask to enable and disable writing of individual bits in the stencil planes. Initially, the mask is all 1's.

glStencilOp(fail, zfail, zpass):

Set stencil test actions

See also

OpenGL Docs

PARAMETERS:

- **fail** (*Enumerated constant*) – Specifies the action to take when the stencil test fails. The initial value is GL_KEEP.
- **zfail** (*Enumerated constant*) – Specifies the stencil action when the stencil test passes, but the depth test fails. zfail accepts the same symboli constants as fail. The initial value is GL_KEEP.
- **zpass** (*Enumerated constant*) – Specifies the stencil action when both the stencil test and the depth test pass, or when the stencil test passes and either there is no depth buffer or depth testing is not enabled. zpass accepts the same symbolic constants as fail. The initial value is GL_KEEP.

glTexCoord (s,t,r,q,v):

B{glTexCoord1d, glTexCoord1f, glTexCoord1i, glTexCoord1s, glTexCoord2d, glTexCoord2f, glTexCoord2i, glTexCoord2s, glTexCoord3d, glTexCoord3f, glTexCoord3i, glTexCoord3s, glTexCoord4d, glTexCoord4f, glTexCoord4i, glTexCoord4s, glTexCoord1dv, glTexCoord1fv, glTexCoord1iv, glTexCoord1sv, glTexCoord2dv, glTexCoord2fv, glTexCoord2iv, glTexCoord2sv, glTexCoord3dv, glTexCoord3fv, glTexCoord3iv, glTexCoord3sv, glTexCoord4dv, glTexCoord4fv, glTexCoord4iv, glTexCoord4sv}

Set the current texture coordinates

See also

OpenGL Docs

PARAMETERS:

- **q** (*s, t, r,*) – Specify s, t, r, and q texture coordinates. Not all parameters are present in all forms of the command.
- **v** (`bgl.Buffer` object. Depends on function prototype. (for 'v' prototypes only)) – Specifies a pointer to an array of one, two, three, or four elements, which in turn specify the s, t, r, and q texture coordinates.

glTexEnv (target, pname, param):

B{glTextEnvf, glTextEnvi, glTextEnvfv, glTextEnviv}

Set texture environment parameters

See also

OpenGL Docs

PARAMETERS:

- **target** (*Enumerated constant*) – Specifies a texture environment. Must be GL_TEXTURE_ENV.
- **pname** (*Enumerated constant*) – Specifies the symbolic name of a single-valued texture environment parameter. Must be GL_TEXTURE_ENV_MODE.
- **param** (*Depends on function prototype.*) – Specifies a single symbolic constant. If function prototype ends in 'v' specifies a pointer to a parameter array that contains either a single symbolic constant or an RGBA color

glTexGen (coord, pname, param):

B{glTexGend, glTexGenf, glTexGeni, glTexGendv, glTexGenfv, glTexGeniv}

Control the generation of texture coordinates

See also

OpenGL Docs

PARAMETERS:

- **coord** (*Enumerated constant*) – Specifies a texture coordinate.
- **pname** (*Enumerated constant*) – Specifies the symbolic name of the texture- coordinate generation function.
- **param** (*Depends on function prototype.*) – Specifies a single-valued texture generation parameter. If function prototype ends in 'v' specifies pointer to an array of texture generation parameters. If pname is GL_TEXTURE_GEN_MODE, then the array must contain a single symbolic constant. Otherwise, params holds the coefficients for the texture-coordinate generation function specified by pname.

glTexImage1D(target, level, internalformat, width, border, format, type, pixels):

Specify a one-dimensional texture image

See also

OpenGL Docs

PARAMETERS:

- **target** (*Enumerated constant*) – Specifies the target texture.
- **level** (*int*) – Specifies the level-of-detail number. Level 0 is the base image level. Level n is the nth mipmap reduction image.
- **internalformat** (*int*) – Specifies the number of color components in the texture.
- **width** (*int*) – Specifies the width of the texture image. Must be 2n+2(border) for some integer n. All implementations support texture images that are at least 64 texels wide. The height of the 1D texture image is 1.
- **border** (*int*) – Specifies the width of the border. Must be either 0 or 1.
- **format** (*Enumerated constant*) – Specifies the format of the pixel data.
- **type** (*Enumerated constant*) – Specifies the data type of the pixel data.
- **pixels** (`bgl.Buffer` object.) – Specifies a pointer to the image data in memory.

glTexImage2D(target, level, internalformat, width, height, border, format, type, pixels):

Specify a two-dimensional texture image

See also

OpenGL Docs

PARAMETERS:

- **target** (*Enumerated constant*) – Specifies the target texture.
- **level** (*int*) – Specifies the level-of-detail number. Level 0 is the base image level. Level n is the nth mipmap reduction image.
- **internalformat** (*int*) – Specifies the number of color components in the texture.
- **width** (*int*) – Specifies the width of the texture image. Must be 2n+2(border) for some integer n. All implementations support texture images that are at least 64 texels wide.
- **height** (*int*) – Specifies the height of the texture image. Must be 2m+2(border) for some integer m. All implementations support texture image that are at least 64 texels high.
- **border** (*int*) – Specifies the width of the border. Must be either 0 or 1.
- **format** (*Enumerated constant*) – Specifies the format of the pixel data.
- **type** (*Enumerated constant*) – Specifies the data type of the pixel data.
- **pixels** (`bgl.Buffer` object.) – Specifies a pointer to the image data in memory.

glTexParameter (target, pname, param):

B{glTexParameterf, glTexParameteri, glTexParameterfv, glTexParameteriv}

Set texture parameters

See also

OpenGL Docs

PARAMETERS:

- **target** (*Enumerated constant*) – Specifies the target texture.
- **pname** (*Enumerated constant*) – Specifies the symbolic name of a single-valued texture parameter.
- **param** (*Depends on function prototype.*) – Specifies the value of pname. If function prototype ends in 'v' specifies a pointer to an array where the value or values of pname are stored.

glTranslate (x, y, z):

B{glTranslatef, glTranslated}

Multiply the current matrix by a translation matrix

See also

OpenGL Docs

PARAMETERS:

z (*x, y,*) – Specify the x, y, and z coordinates of a translation vector.

glViewport(x,y,width,height):

Set the viewport

See also

OpenGL Docs

PARAMETERS:

- **y** (*x,*) – Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0,0).
- **height** (*width,*) – Specify the width and height of the viewport. When a GL context is first attached to a window, width and height are set to t dimensions of that window.

glUseProgram(program):

Installs a program object as part of current rendering state

See also

OpenGL Docs

PARAMETERS:

program (*int*) – Specifies the handle of the program object whose executables are to be used as part of current rendering state.

glValidateProgram(program):

Validates a program object

See also

OpenGL Docs

PARAMETERS:

program (*int*) – Specifies the handle of the program object to be validated.

glLinkProgram(program):

Links a program object.

See also

OpenGL Docs

PARAMETERS:

program (*int*) – Specifies the handle of the program object to be linked.

glActiveTexture(texture):

Select active texture unit.

See also

OpenGL Docs

PARAMETERS:

texture (*int*) – Constant in `GL_TEXTURE0` 0 - 8

glAttachShader(program, shader):

Attaches a shader object to a program object.

See also

OpenGL Docs

PARAMETERS:

- **program** (*int*) – Specifies the program object to which a shader object will be attached.
- **shader** (*int*) – Specifies the shader object that is to be attached.

glCompileShader(shader):

Compiles a shader object.

See also

OpenGL Docs

PARAMETERS:

shader (*int*) – Specifies the shader object to be compiled.

glCreateProgram():

Creates a program object

See also

OpenGL Docs

RETURN TYPE:

int

RETURNS:

The new program or zero if an error occurs.

glCreateShader(shaderType):

Creates a shader object.

See also

OpenGL Docs

PARAMETERS:

shaderType (Specifies the type of shader to be created. Must be one of `GL_VERTEX_SHADER`, `GL_TESS_CONTROL_SHADER`, `GL_TESS_EVALUATION_SHADER`, `GL_GEOMETRY_SHADER`, or `GL_FRAGMENT_SHADER`.)

RETURN TYPE:

int

RETURNS:

0 if an error occurs.

glDeleteProgram(program):

Deletes a program object.

See also

OpenGL Docs

PARAMETERS:

program (*int*) – Specifies the program object to be deleted.

glDeleteShader(shader):

Deletes a shader object.

See also

OpenGL Docs

PARAMETERS:

shader (*int*) – Specifies the shader object to be deleted.

glDetachShader(program, shader):

Detaches a shader object from a program object to which it is attached.

See also

OpenGL Docs

PARAMETERS:

- **program** (*int*) – Specifies the program object from which to detach the shader object.
- **shader** (*int*) – pecifies the program object from which to detach the shader object.

glGetAttachedShaders(program, maxCount, count, shaders):

Returns the handles of the shader objects attached to a program object.

See also

OpenGL Docs

PARAMETERS:

- **program** (*int*) – Specifies the program object to be queried.
- **maxCount** (*int*) – Specifies the size of the array for storing the returned object names.
- **count** (`bgl.Buffer` int buffer.) – Returns the number of names actually returned in objects.
- **shaders** (`bgl.Buffer` int buffer.) – Specifies an array that is used to return the names of attached shader objects.

glGetProgramInfoLog(program, maxLength, length, infoLog):

Returns the information log for a program object.

See also

OpenGL Docs

PARAMETERS:

- **program** (*int*) – Specifies the program object whose information log is to be queried.
- **maxLength** (*int*) – Specifies the size of the character buffer for storing the returned information log.
- **length** (`bgl.Buffer` int buffer.) – Returns the length of the string returned in **infoLog** (excluding the null terminator).

- **infoLog** (`bgl.Buffer` char buffer.) – Specifies an array of characters that is used to return the information log.

glGetShaderInfoLog(program, maxLength, length, infoLog):

Returns the information log for a shader object.

See also

OpenGL Docs

PARAMETERS:

- **shader** (*int*) – Specifies the shader object whose information log is to be queried.
- **maxLength** (*int*) – Specifies the size of the character buffer for storing the returned information log.
- **length** (`bgl.Buffer` int buffer.) – Returns the length of the string returned in **infoLog** (excluding the null terminator).
- **infoLog** (`bgl.Buffer` char buffer.) – Specifies an array of characters that is used to return the information log.

glGetProgramiv(program, pname, params):

Returns a parameter from a program object.

See also

OpenGL Docs

PARAMETERS:

- **program** (*int*) – Specifies the program object to be queried.
- **pname** (*int*) – Specifies the object parameter.
- **params** (`bgl.Buffer` int buffer.) – Returns the requested object parameter.

glIsShader(shader):

Determines if a name corresponds to a shader object.

See also

OpenGL Docs

PARAMETERS:

shader (*int*) – Specifies a potential shader object.

glIsProgram(program):

Determines if a name corresponds to a program object

See also

OpenGL Docs

PARAMETERS:

program (*int*) – Specifies a potential program object.

glGetShaderSource(shader, bufSize, length, source):

Returns the source code string from a shader object

See also

OpenGL Docs

PARAMETERS:

- **shader** (*int*) – Specifies the shader object to be queried.
- **bufSize** (*int*) – Specifies the size of the character buffer for storing the returned source code string.
- **length** (`bgl.Buffer` int buffer.) – Returns the length of the string returned in source (excluding the null terminator).

- **source** (`bgl.Buffer` char.) – Specifies an array of characters that is used to return the source code string.

glShaderSource(shader, shader_string):

Replaces the source code in a shader object.

See also

OpenGL Docs

PARAMETERS:

- **shader** (*int*) – Specifies the handle of the shader object whose source code is to be replaced.
- **shader_string** (*string*) – The shader string.

class bgl.**Buffer**

The Buffer object is simply a block of memory that is delineated and initialized by the user. Many OpenGL functions return data to a C-style pointe however, because this is not possible in python the Buffer object can be used to this end. Wherever pointer notation is used in the OpenGL functio the Buffer object can be used in it's bgl wrapper. In some instances the Buffer object will need to be initialized with the template parameter, while i other instances the user will want to create just a blank buffer which will be zeroed by default.

```
import bgl

myByteBuffer = bgl.Buffer(bgl.GL_BYTE, [32, 32])
bgl.glGetPolygonStipple(myByteBuffer)

print(myByteBuffer.dimensions)
print(myByteBuffer.to_list())

sliceBuffer = myByteBuffer[0:16]
print(sliceBuffer)
```

dimensions

The number of dimensions of the Buffer.

to_list()

The contents of the Buffer as a python list.

__init__(type, dimensions, template = None):

This will create a new Buffer object for use with other bgl OpenGL commands. Only the type of argument to store in the buffer and the dimensions of the buffer are necessary. Buffers are zeroed by default unless a template is supplied, in which case the buffer is initialized to the template.

PARAMETERS:

- **type** (*int*) – The format to store data in. The type should be one of GL_BYTE, GL_SHORT, GL_INT, or GL_FLOAT.
- **dimensions** (*An int or sequence object specifying the dimensions of the buffer.*) – If the dimensions are specified as an int a linear array will be created for the buffer. If a sequence is passed for the dimensions, the buffer becomes n-Dimensional, where n is equal to th number of parameters passed in the sequence. Example: [256,2] is a two- dimensional buffer while [256,256,4] creates a three-dimensional buffer. You can think of each additional dimension as a sub-item of the dimension to the left. i.e. [10,2] is a 10 element array each with 2 sub-items. [(0,0), (0,1), (1,0), (1,1), (2,0), …] etc.
- **template** (*A python sequence object (optional)*) – A sequence of matching dimensions which will be used to initialize the Buffer. If a template is not passed in all fields will be initialized to 0.

RETURN TYPE:

Buffer object

RETURNS:

The newly created buffer as a PyObject.