**OPENGL**

**Primitive types**

**(GLuint, GLFloat, etc)**

**VAO**

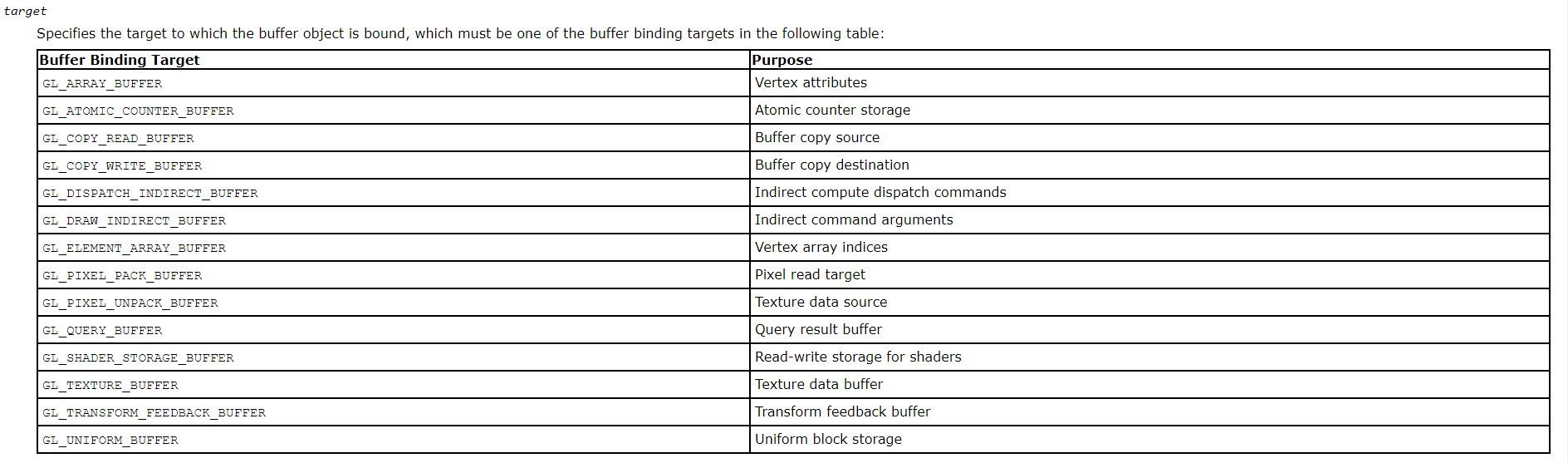
glGenVertexArrays - glGenVertexArrays returns *n* vertex array object names in *arrays*. There is no guarantee that the names form a contiguous set of integers; however, it is guaranteed that none of the returned names was in use immediately before the call to glGenVertexArrays.

glBindVertexArray - glBindVertexArray binds the vertex array object with name *array*. *array* is the name of a vertex array object previously returned from a call to [glGenVertexArrays](https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glGenVertexArrays.xhtml), or zero to break the existing vertex array object binding.

If no vertex array object with name *array* exists, one is created when *array* is first bound. If the bind is successful no change is made to the state of the vertex array object, and any previous vertex array object binding is broken.

**VBO**

glGenBuffers - glGenBuffers returns *n* buffer object names in *buffers*. There is no guarantee that the names form a contiguous set of integers; however, it is guaranteed that none of the returned names was in use immediately before the call to glGenBuffers.

glBindBuffer - 

glBufferData - glBufferData and glNamedBufferData create a new data store for a buffer object. In case of glBufferData, the buffer object currently bound to *target* is used. For glNamedBufferData, a buffer object associated with ID specified by the caller in *buffer* will be used instead.

**Uniforms**

glGetUniformLocation - glGetUniformLocation returns an integer that represents the location of a specific uniform variable within a program object. *name* must be a null terminated string that contains no white space. *name* must be an active uniform variable name in *program* that is not a structure, an array of structures, or a subcomponent of a vector or a matrix. This function returns -1 if *name* does not correspond to an active uniform variable in *program*, if *name* starts with the reserved prefix "gl\_", or if *name* is associated with an atomic counter or a named uniform block.

**GLM**

**vectors**

**matrices**

**transformations -** <https://open.gl/transformations>

**Translate**

**Rotate**

**Scale**

**OpenGL**

**Instance Rendering -** <https://learnopengl.com/Advanced-OpenGL/Instancing>

**Math**

**Vectors**

**2D**

**3D**

**4D**

**3D Shapes**

**Cube**

**Cone**

**Cylinder**

**Tube**

**Sphere**

**Matrices**

**Transformations**

**Scale**

**Translate**

**Rotate**

**Euler**

**Rodriguez**

**Axis**

**Angle**

**Quaternions**

**Quaternion vs matrix**

**Quaternion to matrix**

**Gimbal Lock - Gimbal lock** is the loss of one [degree of freedom](https://en.wikipedia.org/wiki/Degree_of_freedom_(mechanics)) in a three-dimensional, three-[gimbal](https://en.wikipedia.org/wiki/Gimbal) mechanism that occurs when the axes of two of the three gimbals are driven into a parallel configuration, "locking" the system into [rotation](https://en.wikipedia.org/wiki/Rotation) in a degenerate two-dimensional space.

**Vector Spaces -** <https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2018/ENU/Maya-Basics/files/GUID-A63AC5C8-8822-42AC-827E-164B5266DA03-htm.html>

**Projections**

**Cameras**

**Interpolation - SLERP** is a spherical linear interpolation. The interpolation is mapped as though on a quarter segment of a circle so you get the slow out and slow in effect. The distant between each step is not equidistant. **LERP** is a linear interpolation so that the distant between each step is equal across the entire interpolation.

**LERP**

**SLERP**

**Time-based programing (opposed to per frame actions)**

**Physics**

**Bounding Volumes / Rigid Bodies**

**Axis Aligned Bounding Box** An **axis**-aligned **bounding box** (AABB) is a rectangular **box** whose face normals are parallel to the **axes** of the coordinate system.

**Oriented Bounding Box**

**Axis Realigned Bounding Box doesn’t exist - mystery**

**Separation Axis Test -** The Separating Axis Theorem (SAT for short) essentially states if you are able to draw a line to separate two polygons, then they do not collide. It's that simple.

**Spatial Optimization**

**Grids**

**Trees**

**Quadtrees**

**Octrees – a node has eight children**

**Newton’s Laws**

**Inertia**

**Relationship between mass and force**

**Action/Reaction**