PyGLy Documentation

Release 0.0.1

Adam Griffiths

September 01, 2015

1 Concepts **3**
1.1 Transform Spaces 3

2 API **7**
2.1 OpenGL functions 7
2.2 Viewport 7
2.3 View Matrices 7
2.4 Monkey Patch 7
2.5 Transform Spaces 7
2.6 Transform Systems 8
2.7 Tree 8
2.8 Scene Graph 8
2.9 Rendering 9
2.10 Weak Method Reference 10

3 Indices and tables **11**

Python Module Index **13**

Pure Python OpenGL framework

CHAPTER 1

Concepts

1.1 Transform Spaces

See also:

Transform Spaces

1.1.1 Co-ordinate Systems

World Space

The World Space co-ordinate system is the global co-ordinate system in which all objects are located.

When using World Space, we are describing objects in terms of the global origin.

See also:

Inertial Space

Local Space

Local Space co-ordinate systems are relative to another object.

Using Local Space simplify the management and upkeep of 3D objects because we can specify them relative to another object.

Example

Image you have two 3D objects: a Soldier object, and a Gun object.

The Soldier and Gun are placed using co-ordinates in World Space.

If the Soldier picks the gun up, how do we keep the Gun positioned on the Soldier's body?

We can use World Space co-ordinates to place the Gun. To do this we need to know the Soldier's position and the position relative to the Soldier to place the gun. If the Soldier moves, we must re-calculate this value.

Or we could position the Gun using a Local Space co-ordinate system relative to the Soldier. Using this, we only need to know the position relative to the Soldier to place the gun. If the Soldier moves, we don't need to re-calculate anything because the Gun is relative to the Soldier and will move with it.

1.1.2 Transforms

Manipulation of 3D spaces is performed through the use of Transform classes. These provide access to common functions related to translation and orientation of objects in 3D space.

PyGLy provides two types of Transform classes: Transform, and WorldTransform. These objects differ in the co-ordinate systems the functions manipulate.

The Transform class implements Local Space manipulations. Modifications done using the Transform class are performed with no understanding of World Space. When used outside of a hierarchy, the Transform space is relative to World Space. When used inside a hierarchy, the Transform space is relative to the parent's Local Space.

The WorldTransform class implements World Space manipulations. A WorldTransform object is linked to a Transform object. Modifications done using the WorldTransform class are performed in World Space. This can be used to set absolute positioning of an object despite it's parent's transform.

The WorldTransform class implements the TreeNode class which adds the concept of a hierarchy. WorldTransforms can be joined together in a Parent - Child tree (1 parent maximum). This enables the inheritance of parent transforms.

Usage of WorldTransform or Transform objects is not mutually exclusive. Modifications performed on the World-Transform or the linked Transform object are propagated to each other.

1.1.3 Transform Spaces

Each Transform class exposes two co-ordinate spaces, Object Space, and Inertial Space.

When using Object Space, the X,Y and Z axis inherit the orientation from the Object. As the Object's orientation changes, the X,Y and Z axis presented by functions change accordingly.

When using Inertial Space, the X,Y and Z axis remain fixed regardless of the Object's orientation.

Object Space

The Object Space co-ordinate system's axis are fixed to the object's orientation. As the object is rotated, the Object Space axis will change with the object.

See also:

Object Space

Inertial Space

The Inertial Space co-ordinate system orientation is relative to it's parent orientation. When there is no parent, the co-ordinate system is the same as the global World Space. The Inertial Space axis does not change when the object is rotated.

See also:

Inertial Space

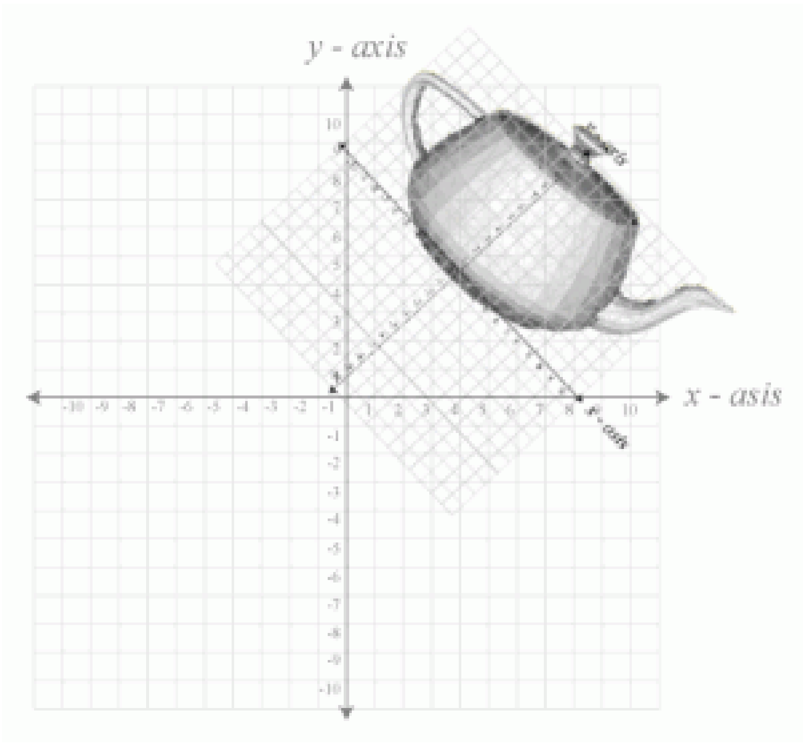

Fig. 1.1: *Object Transform Space.*

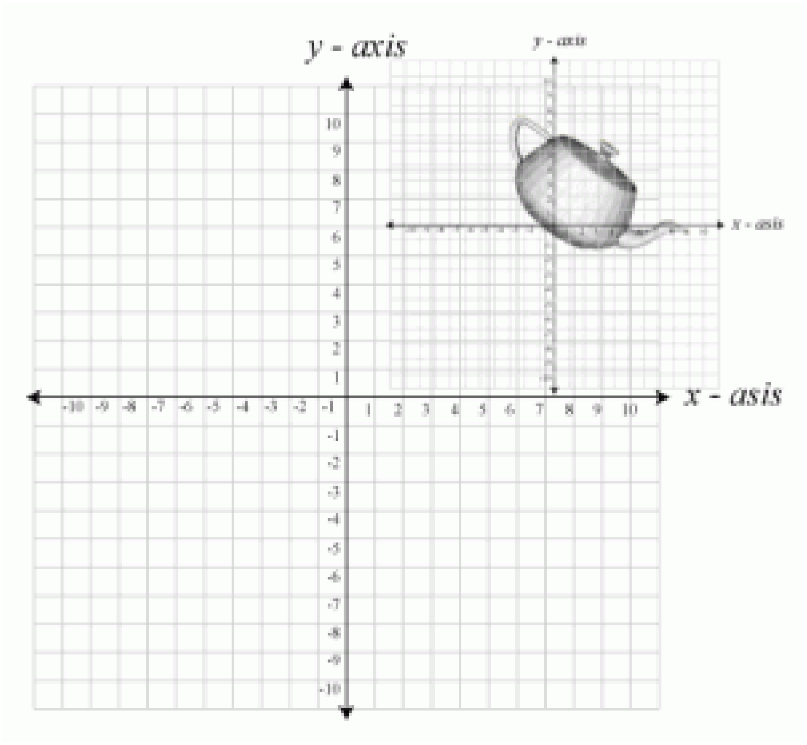

Fig. 1.2: *Inertial Transform Space*

CHAPTER 2

API

2.1 OpenGL functions

2.1.1 Common Functions

2.1.2 Common Utilities

2.1.3 Legacy

2.2 Viewport

2.2.1 Viewport

2.2.2 Ratio Viewport

See also:

Module `pygly.gl` Documentation of the `pygly.gl` module. Contains viewport controls.

2.3 View Matrices

2.3.1 View Matrix

2.3.2 Perspective View Matrix

2.3.3 Orthogonal View Matrix

2.4 Monkey Patch

2.5 Transform Spaces

See also:

Transform Systems

2.5.1 Object Space

2.5.2 Inertial Space

2.6 Transform Systems

See also:

Transform Spaces Transform Spaces

2.6.1 Transform

2.6.2 World Transform

2.7 Tree

2.7.1 Tree Node

2.7.2 Tree Leaf

class `pygly.tree_leaf.`**`TreeLeaf`**

Base class for Tree Leaf objects.

Supports a single parent. Cannot have children.

`__init__``()`

Creates a tree leaf object.

`parent`

The current parent of the node or None if there isn't one.

This is an @property decorated method which allows retrieval and assignment of the scale value.

`predecessors``()`

Returns successive parents of the node.

Generator function that allows iteration up the tree.

2.8 Scene Graph

2.8.1 Scene Node

2.8.2 Camera Node

2.8.3 Render Node

2.8.4 Render Callback Node

See also:

Class `pygly.render_node.RenderNode` Documentation of the `pygly.render_node.RenderNode` class, the parent of this class.

2.9 Rendering

2.9.1 OpenGL

2.9.2 Shaders

2.9.3 Buffers

class `pygly.vertex_array.`**`VertexArray`**

Wraps OpenGL Vertex Array Objects.

Provides wrappers around standard functions and higher level wrappers with PyGLy.BufferRegion interfaces.

Example:

```
vs = Shader( GL_VERTEX_SHADER, shader_source['vert'] )
fs = Shader( GL_FRAGMENT_SHADER, shader_source['frag'] )
shader = ShaderProgram( vs, fs )

# a basic triangle
vertices = numpy.array(
    [
        #   X    Y    Z          R    G    B
        (( 0.0, 1.0, 0.0),     (1.0, 0.0, 0.0)),
        ((-2.0,-1.0, 0.0),     (0.0, 1.0, 0.0)),
        (( 2.0,-1.0, 0.0),     (0.0, 0.0, 1.0)),
        ],
    dtype = [
        ('position','float32',(3,)),
        ('colour','float32',(3,))
        ]
    )
buffer = DtypeVertexBuffer(
    vertices.dtype,
    GL_ARRAY_BUFFER,
    GL_STATIC_DRAW,
    data = vertices
    )

vao = VertexArray()

vao.bind()
buffer.bind()
buffer.set_attribute_pointer_dtype( shader, 'in_position', 'position' )
buffer.set_attribute_pointer_dtype( shader, 'in_colour', 'colour' )
buffer.unbind()
vao.unbind()
```

Warning: This is an OpenGL Core function (>=3.0) and should not be called for Legacy profile applications (<=2.1).

`__init__``()`

`bind``()`

`handle`

`unbind``()`

2.9.4 Texturing

2.9.5 Sorting

2.10 Weak Method Reference

Provides an implementation of a WeakMethodReference for weak references to functions and methods.

The standard weakref module in Python cannot store references to non-bound functions and should not be used to perform this task.

Code borrowed from the following places: http://code.activestate.com/recipes/81253/ http://stackoverflow.com/questions/3942303/how-does-a-python-set-check-if-two-objects-are-equal-what-methods-does-an-o

class `pygly.weak_method_reference.`**`WeakMethodReference`**(*function=None*)

Provides the ability to store a weak pointer to class members on top of the existing weakref functionality provided by python.

This class also provides comparison operators to allow proper usage in containers such as set([]).

The ability to change the weak reference is not supported to prevent mutability. This is important for container support as the object hash would change after storing it.

`__init__`(*function=None*)

Initialises the weak reference with a function or class method.

Args: function: The object to store a weak reference to. This can be a class, object, method or function.

`is_alive`()

Check if the referenced object is valid.

The equivalent to 'not is_dead()' Make a positive method call because double negatives suck

`is_dead`()

Check if the referenced object is invalid.

Returns: True if the referenced callable was a bound method and the instance no longer exists. Otherwise, return False.

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

pygly.tree_leaf, 8
pygly.vertex_array, 9
pygly.weak_method_reference, 10

Symbols

__init__() (pygly.tree_leaf.TreeLeaf method), 8
__init__() (pygly.vertex_array.VertexArray method), 9
__init__() (pygly.weak_method_reference.WeakMethodReference method), 10

B

bind() (pygly.vertex_array.VertexArray method), 9

H

handle (pygly.vertex_array.VertexArray attribute), 9

I

is_alive() (pygly.weak_method_reference.WeakMethodReference method), 10
is_dead() (pygly.weak_method_reference.WeakMethodReference method), 10

P

parent (pygly.tree_leaf.TreeLeaf attribute), 8
predecessors() (pygly.tree_leaf.TreeLeaf method), 8
pygly.tree_leaf (module), 8
pygly.vertex_array (module), 9
pygly.weak_method_reference (module), 10

T

TreeLeaf (class in pygly.tree_leaf), 8

U

unbind() (pygly.vertex_array.VertexArray method), 9

V

VertexArray (class in pygly.vertex_array), 9

W

WeakMethodReference (class in pygly.weak_method_reference), 10