Matrix and Vector Math

Modern OpenGL depends on matrixes for projection, translation, and other things. To provide out-of-the-box functionality, pyglet's `math` module includes the necessary Matrix and Vector types to cover common use cases. Provided types are: `Vec2`, `Vec3`, `Vec4`, `Mat3`, and `Mat4`.

See the `math` API documentation for a full list of methods and operations that these objects support.

Note

Matrix and Vector types are `tuple` subclasses. They are therefore immutable - all operations return a **new** object; they are not updated in-place.

Creating a Matrix

A Matrix can be created with no arguments, or by passing the initial values:

```
my_matrix = Mat4()
# or
my_matrix = Mat4(1.0, 0.0, 0.0, 0.0,
                 0.0, 1.0, 0.0, 0.0,
                 0.0, 0.0, 1.0, 0.0,
                 0.0, 0.0, 0.0, 1.0)
```

If no arguments are given on creation, an "identity" matrix will be created by default. (1.0 on the main diagonal, as shown here).

More conveniently, you can also create matrixes from one of their helper class methods.

Matrix Helper Methods

When working with 3D graphics in OpenGL, A common operation is setting a perspective projection matrix. The `Mat4` class includes a helper method for this task. The arg similar to what you will find in popular OpenGL math libraries:

latest

;

```
# Perspective (3D) projection matrix:
projection = Mat4.perspective_projection(aspect_ratio, z_near=0.1, z_far=255, fov=60)
```

For setting a projection matrix on the current OpenGL context, pyglet Windows have a `projection` property. This points to a projection matrix used by all of pyglet's built-in classes, and can be set as follows:

```
my_matrix = Mat4.perspective_projection(aspect_ratio, z_near=0.1, z_far=255)
window.projection = my_matrix
```

For working with 2D graphics, a helper method also exists for creating orthogonal projection matrixes, which is created and set in a similar way:

```
# Orthographic (2D) projection matrix:
my_matrix = Mat4.orthogonal_projection(0, width, 0, height, z_near=-255, z_far=255)
window.projection = my_matrix
```

By default, pyglet is already set up with an orthographic projection matrix. It automatically updates this 2D matrix whenever a Window is created or resized. This is done inside of the Window's `on_resize` event handler. If you plan to set your own 3D or 2D projection matrix, a useful pattern is to override the default `on_resize` event handler with your own:

```
@window.event
def on_resize(width, height):
    # window.viewport = (0, 0, width, height)
    window.projection = Mat4.perspective_projection(window.aspect_ratio, z_near=0.1,
z_far=255)
    return pyglet.event.EVENT_HANDLED   # Don't call the default handler
```

In addition to the `projection` property, pyglet Windows also contain a `view` property that points to another matrix that is used internally by pyglet's built-in classes. The view matrix gets multiplied with the projection matrix, controlling the final positioning and rendering of pyglet's built-in objects like Sprites, Shapes, and text. Unlike the projection matrix the view matrix defaults to an indentity matrix, and therefore has no effect initially. This makes it useful for users to set their own matrix on it, and controll zooming or translating (panning) a "camera". The `Mat4` class contains helper methods for creating these types of matrixes:

latest

```
# translate the camera +100 on the X axis:
view_matrix = Mat4.from_translation(vector=Vec3(x=100, y=0, z=0))
# or scale by 2X (be careful of the Z values):
view_matrix = Mat4.from_scale(vector=Vec3(x=2.0, y=2.0, z=1.0))

window.view = view_matrix
```

You can of course create multiple matrixes and multiply them together, such as to zoom and then translate, or translate and then zoom, etc. Be careful of multiplication order, as matrix multiplication is non-commutative.

See the `math` API documentation for a full list of helper class methods.

Matrix Multiplication

Matrix classes in pyglet use the Python `matmul` (`@`) operator for matrix multiplication. The `star` operator (`*`) is not allowed, and will raise an exception. For example:

```
new_matrix = rotation_matrix @ translation_matrix
```

Creating Vectors

pyglet includes classes for 2D, 3D, and 4D vectors: `Vec2`, `Vec3` and `Vec4`. These Vec types support most mathematical operations, and implement helper methods for most common vector operations. Vecs are created with their values.

> myvec2 = Vec2() # same as Vec(x=0.0, y=0.0) myvec2 = Vec2(-4.5, 3.0) myvec3 = Vec3(100.0, 400.0, 1.0)

Vector Operations

TBD