CGL Vectors Library

What is this library?

CGL vectors library is a built-in library of CGL that you can use & utilize for your projects. It performs most of the basic vector & matrix mathematics, and is heavily used in our reference solution.

Why you should use it?

The vectors library contains fully overloaded operators (so you can write vector equations just like normal ones), and it contains a lot of optimizations. These features reduces code size and risk of getting something wrong in the equation. This usually also improves the performance.

References

Data Types

Vector Types

- `Vector2D`
- `Vector3D`
  - `Spectrum` is an alias of `Vector3D`, as RGB vectors are still three-component vectors
- `Vector4D`

Matrix Types

- `Matrix3x3`
- `Matrix4x4`

Constructor

`Vector2D()` / `Vector3D()` / `Vector4D()` Creates a 2D/3D/4D *zero* vector.

`Vector2D(x)` / `Vector3D(x)` / `Vector4D(x)` Creates a 2D/3D/4D vector containing `x`: {x, x} / {x, x, x} / {x, x, x, x}

`Vector2D(x,y)` / `Vector3D(x,y,z)` / `Vector4D(x,y,z,w)` Creates a 2D/3D/4D vector: {x, y} / {x, y, z} / {x, y, z, w}

Operations

Vector Indexing

```
Vector2D v;

x = v.x = v[0]
y = v.y = v[1]
```

For 3D vectors, as colors and spectrums can also be represented, you can also index 3D vectors / spectrums using `r`, `g`, and `b`

```
Vector3D v;

x = v.x = v[0] = r = v.r
y = v.y = v[1] = g = v.g
z = v.z = v[2] = b = v.b
```

For 4D vectors, as colors with transparency also be represented, you can also index 4D vectors / spectrums using `r`, `g`, `b`, and `a`

```
Vector4D v;

x = v.x = v[0] = r = v.r
y = v.y = v[1] = g = v.g
z = v.z = v[2] = b = v.b
w = v.w = v[3] = a = v.a
```

Matrix Indexing

```
MatrixND m
```

`m[n]` is the n-th column of `m`, in a form of a `VectorND` `m[n].x` is the n-th column first row of `m` `m[n][i]` is the n-th column i-th row of `m`

Vector-Scalar Operations

Assume `v` is a vector, `s` is a scalar:

```
VectorND v;
double s;
```

Vector-scalar multiplication / division `v * s` Returns `{v.x * s, v.y * s, v.z * s}` `s * v` Returns `{v.x * s, v.y * s, v.z * s}` `v / s` Returns `{v.x / s, v.y / s, v.z / s}` `s / v` Returns `{s / v.x, s / v.y, s / v.z}`

Vector-Vector Operations

Assume `v1` and `v2` are vectors of the same size.

Vector-vector division (element-wise division) `v1 / v2` returns `{v1.x / v2.x, v1.y / v2.y, v1.z / v2.z}`

Vector-vector addition (element-wise addition) `v1 + v2` returns `{v1.x + v2.x, v1.y + v2.y, v1.z + v2.z}`

Vector-vector subtraction (element-wise subtraction) `v1 - v2` returns `{v1.x - v2.x, v1.y - v2.y, v1.z - v2.z}`

Dot product `dot(v1, v2)` returns dot product `v1.x * v2.x + v1.y * v2.y + v1.z * v2.z`

Cross product `cross(v1, v2)` returns the cross product of `v1` and `v2`

Outer product `outer(v1, v2)` retruns a `MatrixNxN`, the outer product of `v1` and `v2`

Vector Methods `v.rcp()` returns per-entry reciprocal `{1.0 / v.x, 1.0 / v.y, 1.0 / v.z}` `v.norm()` returns euclidean length `sqrt(v.x * v.x + v.y * v.y + v.z * v.z)` `v.norm2()` returns square of euclidean length `v.x * v.x + v.y * v.y + v.z * v.z` `v.unit()` returns normalized unit vector `{v.x / v.norm(), v.y / v.norm(), v.z / v.norm()}` `v.normalize()` normalizes the vector to unit vector. (does not return anything)

For 3D vectors: `v.illum()` returns the perceived brightness of a spectrum (color) vector `v.toColor()` returns a `Color` object from the spectrum object. `Vector3D::fromColor(c)` returns a `Vector3D` object construted from `Color` object `c`.

Matrix-Matrix Operations

Assume `A1` and `A2` are `MatrixNxN`

`A1 - A2` returns element-wise subtraction `A1 + A2` returns element-wise addition `A1 * A2` returns matrix-matrix multiplication

Matrix-Vector Operations

Assume `A` is `MatrixNxN` and `v` is `VectorND`

`A*v` returns matrix-vector multiplication. Returns a `VectorND`

Matrix-Scalar Operations

Assume `A` is `MatrixNxN` and `s` is `scalar`

`A * s` or `s * A` returns `{A[0] * s, A[1] * s, A[2] * s}`

Matrix Methods

`A.det()` retruns determinant of `A` (double) `A.norm()` returns Frobenius norm of `A` `A.inv()` returns the inverse of `A`

Printing Vectors

You can use `std::cout << v << std::endl` to print vectors directly.

You may see it printed out as a color with R,G,B channels, as color is just like any other three-component vector. The R/G/B channels correspond to X/Y/Z axes

Printing Matrices

You can use `std::cout << A << std::endl` to print matrices directly.