Práctica 3: Transformaciones 3D, vectores y matrices

Profesor: Pedro Xavier Contla Romero
Ayudante: Melissa Méndez Servín
Ayudante de laboratorio: Joel Espinosa Longi

Fecha de entrega: 16 de marzo de 2020

Transformaciones 3D

En 3D al igual que en 2D, tenemos las transformaciones de escalamiento, rotación y traslación.

Siendo un escalamiento de la siguiente manera:

$$x' = x * sx$$
$$y' = y * sy$$
$$z' = z * sz$$

La transformación de rotación en tres dimensiones se suele expresar como tres transformaciones, una cada uno de los ejes del espacio 3D, por ejemplo, para una rotación respecto al eje X, tenemos:

$$x' = x$$
$$y' = y \cdot cos(\theta) - z \cdot sen(\theta)$$
$$z' = y \cdot sen(\theta) + z \cdot cos(\theta)$$

Para una rotación respecto al eje Y, sería:

$$x' = x \cdot cos(\theta) + z \cdot sen(\theta)$$
$$y' = y$$
$$z' = -x \cdot sen(\theta) + z \cdot cos(\theta)$$

Y una rotación en el eje Z, es:

$$x' = x \cdot cos(\theta) - y \cdot sen(\theta)$$
$$y' = x \cdot sen(\theta) + y \cdot cos(\theta)$$
$$z' = z$$

La transformación de traslación en tres dimensiones es similar a la de dos dimensiones, solo agregando el factor de desplazamiento para el eje ***Z***:

$$x' = x + tx$$
$$y' = y + ty$$
$$z' = z + tz$$

Todas estas transformaciones, al igual que sus contrapartes en 2D, pueden ser descritas utilizando matrices con una dimensión más que la dimensión del espacio, es decir, para representar las transformaciones en 3D, necesitamos utilizar matrices de $\mathbf{4} \times \mathbf{4}$. Y para representar y operar los vértices de los objetos geométricos en 3D, necesitamos utilizar coordenadas homogéneas, es decir, aumentar una dimensión a la especificación de los vectores, teniendo con esto vectores de 4 dimensiones.

Utilizando lo que han visto en clase, van a implementar las clases: `Vector4` y `Matrix4`, cada una de ellas definidas en su propio archivo javascript: `Vector4.js` y `Matrix4.js`, respectivamente. De forma similar a su práctica 2, las clases serán utilizadas como módulos de JavaScript, por lo que es necesario que utilicen `export` para declarar las clases.

```
export default class Vector4 {
// aquí va el código de la clase Vector4

}
```

Y desde otros archivos se utiliza un `import` de la siguiente manera:

```
import Vector3 from "./Vector3.js";
```

Vector4

La clase `Vector4` representa vectores de cuadro componentes, x, y, z y w. Los cuales serán utilizados para representar vértices de objetos geométricos en tres dimensiones.

El constructor recibe 4 parámetros numéricos y crea un objeto que representa un vector de cuatro componentes (`constructor(x, y, z, w)`), en caso de no recibir valores en los argumentos, devuelve el vector $(0, 0, 0, 0)$.

```
/**
 * @param {Vector4} u
 * @param {Vector4} v
 * @return {Vector4}
 */
static add(u, v)
```

`add` es una función que devuelve la suma de sus argumentos.

```
/**
 * @return {Vector4}
 */
clone()
```

`clone` es una función que devuelve un objeto el cual contiene los mismos valores que el objeto desde el cual se invocó la función.

```
/**
 * @param {Vector4} u
 * @param {Vector4} v
 * @return {Number}
 */
static distance(u, v)
```

`distance` es una función que devuelve la distancia euclidiana que hay entre sus argumentos.

```
/**
 * @param {Vector4} u
 * @param {Vector4} v
 * @return {Number}
 */
static dot(u, v)
```

`dot` es una función que devuelve el producto punto de sus argumentos.

```
/**
 * @param {Vector4} u
 * @param {Vector4} v
 * @return {Boolean}
 */
static equals(u, v)
```

`equals` es una función que devuelve verdadero en caso de que sus argumentos sean aproximadamente iguales (con una $\varepsilon = 0.000001$) y falso en caso contrario.

```
/**
 * @param {Vector4} u
 * @param {Vector4} v
 * @return {Boolean}
 */
static exactEquals(u, v)
```

`exactEquals` es una función que devuelve verdadero en caso de que sus argumentos sean exactamente iguales y falso en caso contrario.

```
/**
 * @return {Vector4}
 */
normalize()
```

`normalize` es una función que devuelve el vector resultado de la normalización del vector que invoca la función.

```
/**
 * @param {Number} x
 * @param {Number} y
 * @param {Number} z
 */
set(x, y, z)
```

`set` es una función que asigna nuevos valores a los componentes del vector con que se llama.

```
/**
 * @param {Vector4} u
 * @param {Vector4} v
 * @return {Number}
 */
static squaredDistance(u, v)
```

`squaredDistance` es una función que devuelve la distancia euclidiana al cuadrado que hay entre sus argumentos.

```
/**
 */
zero()
```
`zero` es una función que asigna cero a cada componente del vector que invoca la función.

Matrix4

La clase `Matrix4` representa matrices de 4×4. Y nos permitirán representar y construir las transformaciones en tres dimensiones.

El constructor recibe 16 parámetros numéricos, `Matrix4(a00, a01, a02, a03, a10, a11, a12, a13, a20, a21, a22, a23, a30, a31, a32, a33)` y construye la siguiente matriz:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

En caso de no recibir valores en los argumentos, devuelve la matriz identidad:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
/**
 * @param {Matrix4} m1
 * @param {Matrix4} m2
 * @return {Matrix4}
 */
static add(m1, m2)
```
`add` es una función que devuelve la suma dos matrices.

```
/**
 * @return {Matrix4}
 */
adjoint()
```
`adjoint` es una función que devuelve la matriz adjunta (o matriz de cofactores), de la matriz con que se invoca la función.

```
/**
 * @return {Matrix4}
 */
clone()
```
`clone` es una función que devuelve un objeto el cual contiene los mismos valores que el objeto desde el cual se invocó la función.

```
/**
 * @return {Number}
 */
determinant()
```
`determinant` es una función que devuelve el determinante de la matriz.

```
/**
 * @param {Matrix4} m1
 * @param {Matrix4} m2
 * @return {Boolean}
 */
static equals(m1, m2)
```
`equals` es una función que devuelve verdadero en caso de que sus argumentos sean aproximadamente iguales (con una $\varepsilon = \mathbf{0.000001}$) y falso en caso contrario.

```
/**
 * @param {Matrix4} m1
 * @param {Matrix4} m2
 * @return {Boolean}
 */
static exactEquals(m1, m2)
```

`exactEquals` es una función que devuelve verdadero en caso de que sus argumentos sean exactamente iguales y falso en caso contrario.

```
/**
 * @param {Number} left
 * @param {Number} right
 * @param {Number} bottom
 * @param {Number} top
 * @param {Number} near
 * @param {Number} far
 * @return {Matrix4}
 */
static frustum(left, right, bottom, top, near, far)
```

`frustum` es una función que construye una matriz que representa la pirámide truncada (*view frustum*), determinada por los planos dados por los parámetros `left`, `right`, `bottom`, `top`, `near` y `far`.

```
/**
 */
identity()
```

`identity` es una función que asigna los valores de la matriz identidad a la matriz desde donde se invocó la función.

```
/**
 * @return {Matrix4}
 */
invert()
```

`invert` es una función que devuelve la matriz inversa de la matriz con la que se invocó la función.

```
/**
 * @param {Vector3} eye
 * @param {Vector3} center
 * @param {Vector3} up
 * @return {Matrix4}
 */
static lookAt(eye, center, up)
```

`lookAt` es una función que devuelve la matriz de vista a partir de la posición del ojo (`eye`) el centro de interés (`center`) y el vector hacia arriba (`up`).

```
/**
 * @param {Matrix4} m1
 * @param {Matrix4} m2
 * @return {Matrix4}
 */
static multiply(m1, m2)
```
`multiply` es una función que devuelve la multiplicación de dos matrices.

```
/**
 * @param {Matrix4} m1
 * @param {Number} c
 * @return {Matrix4}
 */
static multiplyScalar(m1, c)
```
`multiplyScalar` es una función que devuelve una matriz que es el resultado de multiplicar cada componente por un escalar.

```
/**
 * @param {Vector4} v
 * @return {Vector4}
 */
multiplyVector(v)
```
`multiplyVector` es una función que devuelve el vector resultado de multiplicar el vector `v` por la matriz con que se llama la función. Esta función es la que se va a llamar cuando se apliquen las transformaciones sobre los vectores.

```
/**
 * @param {Number} left
 * @param {Number} right
 * @param {Number} bottom
 * @param {Number} top
 * @param {Number} near
 * @param {Number} far
 * @return {Matrix4}
 */
static ortho(left, right, bottom, top, near, far)
```
`ortho` es una función que devuelve una matriz que corresponde a una proyección ortogonal, determinada por los planos dados por los parámetros `left`, `right`, `bottom`, `top`, `near` y `far`.

```
/**
 * @param {Number} fovy
 * @param {Number} aspect
 * @param {Number} near
 * @param {Number} far
 * @return {Matrix4}
 */
static perspective(fovy, aspect, near, far)
```

`perspective` es una función que devuelve una matriz que corresponde a una proyección en perspectiva. El parámetro `fovy` corresponde al campo de visión vertical (*field of view*), el parámetro `aspect` corresponde a la relación de aspecto, `near` es la distancia del plano más cercano y `far` es la distancia del plano más lejano.

```
/**
 * @param {Number} theta
 * @return {Matrix4}
 */
static rotateX(theta)
```

`rotateX` es una función que devuelve una matriz de rotación en 3D sobre el eje $\boldsymbol{X}$ con el ángulo (en radianes) dado por el parámetro `rad`.

```
/**
 * @param {Number} theta
 * @return {Matrix4}
 */
static rotateY(theta)
```

`rotateY` es una función que devuelve una matriz de rotación en 3D sobre el eje $\boldsymbol{Y}$ con el ángulo (en radianes) dado por el parámetro `rad`.

```
/**
 * @param {Number} theta
 * @return {Matrix4}
 */
static rotateZ(theta)
```

`rotateZ` es una función que devuelve una matriz de rotación en 3D sobre el eje $\boldsymbol{Z}$ con el ángulo (en radianes) dado por el parámetro `rad`.

```
/**
 * @param {Vector3} v
 * @return {Matrix4}
 */
static scale(v)
```

`scale` es una función que devuelve una matriz de escalamiento en 3D con los factores de escala determinados por las componentes del vector `v`.

```
/**
 * @param {Number} a00
 * @param {Number} a01
 * @param {Number} a02
 * @param {Number} a03
 * @param {Number} a10
 * @param {Number} a11
 * @param {Number} a12
 * @param {Number} a13
 * @param {Number} a20
 * @param {Number} a21
 * @param {Number} a22
 * @param {Number} a23
 * @param {Number} a30
 * @param {Number} a31
 * @param {Number} a32
 * @param {Number} a33
 */
set(a00, a01, a02, a03, a10, a11, a12, a13, a20, a21, a22, a23, a30, a31, a32, a33)
```

`set` es una función que asigna nuevos valores a los componentes de la matriz con que se llama.

```
/**
 * @param {Matrix4} m1
 * @param {Matrix4} m2
 * @return {Matrix4}
 */
static subtract(m1, m2)
```

`subtract` es una función que sustrae componente a componente la matriz `m2` de la matriz `m1`.

```
/**
 * @param {Vector3} v
 * @return {Matrix4}
 */
static translate(v)
```

`translate` es una función que devuelve una matriz de traslación en 3D con los factores de desplazamiento dados por las componentes del vector `v`.

```
/**
 * @return {Matrix4}
 */
transpose()
```

`transpose` es una función que devuelve la matriz transpuesta de la matriz desde donde se invocó la función.

Notas de entrega

- Recuerden que para que funcionen los módulos de JavaScript deben utilizar un servidor local. Como les comente en el laboratorio mi recomendación es que utilicen nodejs y http-server.
- Para la entrega deben subir al Classroom un archivo zip con su número de cuenta como nombre de archivo (`NUMCUENTA.zip`).
- Si realizan la práctica en pareja, dentro del zip de la entrega se debe agregar un archivo `README.txt` que contenga el nombre de los integrantes del equipo. Y ambos deben subir el archivo zip como entrega.
- Podrán entregar la práctica después de la fecha establecida para la entrega (16 de marzo de 2020), pero por cada día de retraso se penalizará con un punto menos. Por este motivo la fecha de entrega es inamovible.
- El código debe estar documentado, en caso contrario se penalizará la calificación.