Práctica 4: Objetos geométricos 3D con WebGL

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

Fecha de entrega: 22 de abril de 2020

En los ejemplos:

- 077 - varios prismas rectangulares,
- 078 - tetraedro,
- 079 - octaedro,
- 080 - dodecaedro,
- 081 - icosaedro,
- 082 - esfera,
- 083 - cono,
- 084 - cilindro y
- 085 - toro

Se presentan diversas figuras geométricas tridimensionales, los ejemplos están desarrollados directamente sobre y el canvas, así que su práctica consiste en programar las mismas figuras, pero listas para ser utilizadas en sus programas de WebGL. Van a crear un archivo JavaScript para cada una de las figuras:

- `PrismaRectangular.js`,
- `Tetraedro.js`,
- `Octaedro.js`,
- `Dodecaedro.js`,
- `Icosaedro.js`,
- `Esfera.js`,
- `Cono.js`,
- `Cilindro.js` y
- `Toro.js`;

y dentro de cada clase van a definir el código correspondiente. Por ejemplo:

```
export default class PrismaRectangular {
  // aquí va el código de la clase prisma rectangular
}
```

Cada clase tendrá un constructor con los mismos parámetros que las funciones `build_ALGO_` de los ejemplos de figuras geométricas, más dos parámetros extra, el parámetro `gl` como primer argumento, que corresponde a una referencia al contexto de render de WebGL (la cual es necesaria para crear los buffers de datos) y el parámetro `color` que será el color del objeto especificado como un arreglo de cuatro elementos, `[R, G, B, A]`, (hay que considerar que `A=1` es un color opaco y `A=0` es completamente transparente) con valores en el intervalo `[0, 1]`. Por ejemplo, en el código del prisma rectangular se tiene la siguiente función:

```
buildBox(width, height, length, initial_transform),
```

entonces el constructor del prisma rectangular sería el siguiente:

```
export default class PrismaRectangular {
  /**
   * @param {WebGLRenderingContext} gl
   * @param {Number[]} color
   * @param {Number} width
   * @param {Number} height
   * @param {Number} length
   * @param {Matrix4} initial_transform
  */
  constructor(gl, color, width, height, length, initial_transform) {
    // definición del constructor del prisma rectangular
  }
}
```

Cada objeto geométrico tendrá una función para dibujarse

```
/**
  * @param {WebGLRenderingContext} gl
  * @param {GLint} positionAttributeLocation
  * @param {WebGLUniformLocation} colorUniformLocation
  * @param {WebGLUniformLocation} PVM_matrixLocation
  * @param {Matrix4} projectionViewMatrix
  */
draw(gl, positionAttributeLocation, colorUniformLocation, PVM_matrixLocation,
     projectionViewMatrix)
```

`draw` es una función que dibuja el objeto utilizando el contexto de render que recibe cómo parámetro (`gl`), una referencia al `attribute vec4 a_position` para enviar la información de la posición de los vértices del objeto geométrico al GPU (`positionAttributeLoaction`), una referencia al `uniform vec4 u_color` para enviar la información del color del objeto (`colorUniformLocation`), una referencia al `uniform mat4 u_PVM_matrix` para enviar información sobre la matriz que concatena las transformaciones de perspectiva, vista y modelo (`PVM_matrixLocation`) y la información de la matriz de proyección y vista (`projectionViewMatrix`).

Además deben agregar a la clase `Matrix4` la siguiente función:

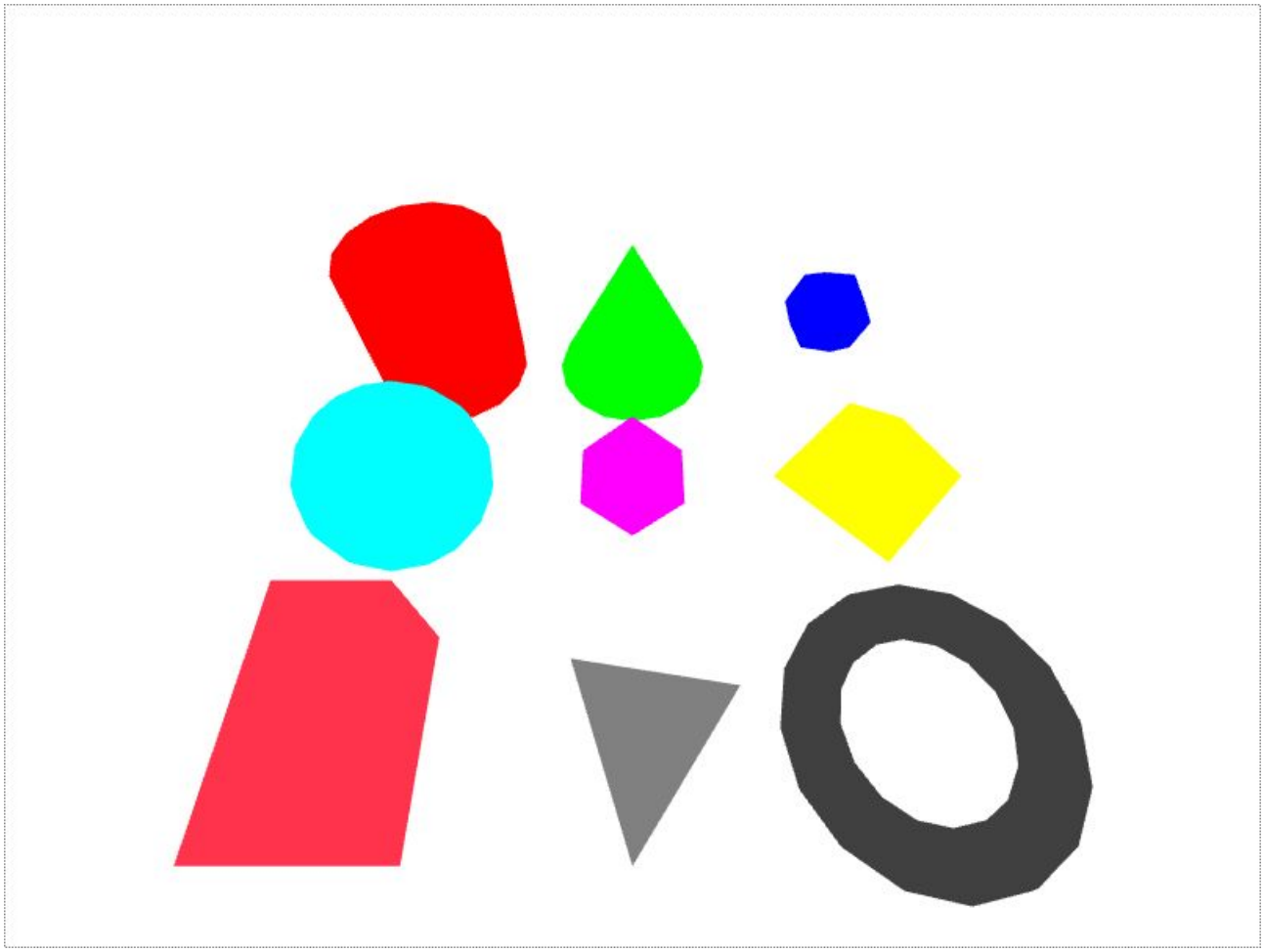
```
/**
  * @return {Array}
  */
toArray()
```

`toArray` es una función que convierte la matriz en un arreglo, de tal forma que si tenemos la siguiente matriz:

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

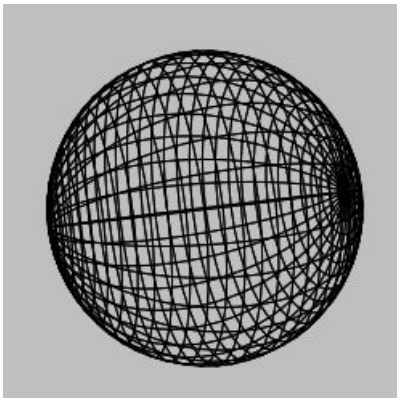
donde `a03`, `a13`, `a23` y `a33` tienen la información de translación, entonces la función devuelve el arreglo: `[a00, a10, a20, a30, a01, a11, a21, a31, a02, a12, a22, a32, a03, a13, a23, a33]`. Esta función es necesaria para pasarle a WebGL la información de las matrices de transformación.

Junto con el archivo pdf de la práctica se adjuntan los archivos `index.html` y `Main.js`, como archivos de prueba a partir de los cuales deben obtener la siguiente imagen.



Notas de implementación

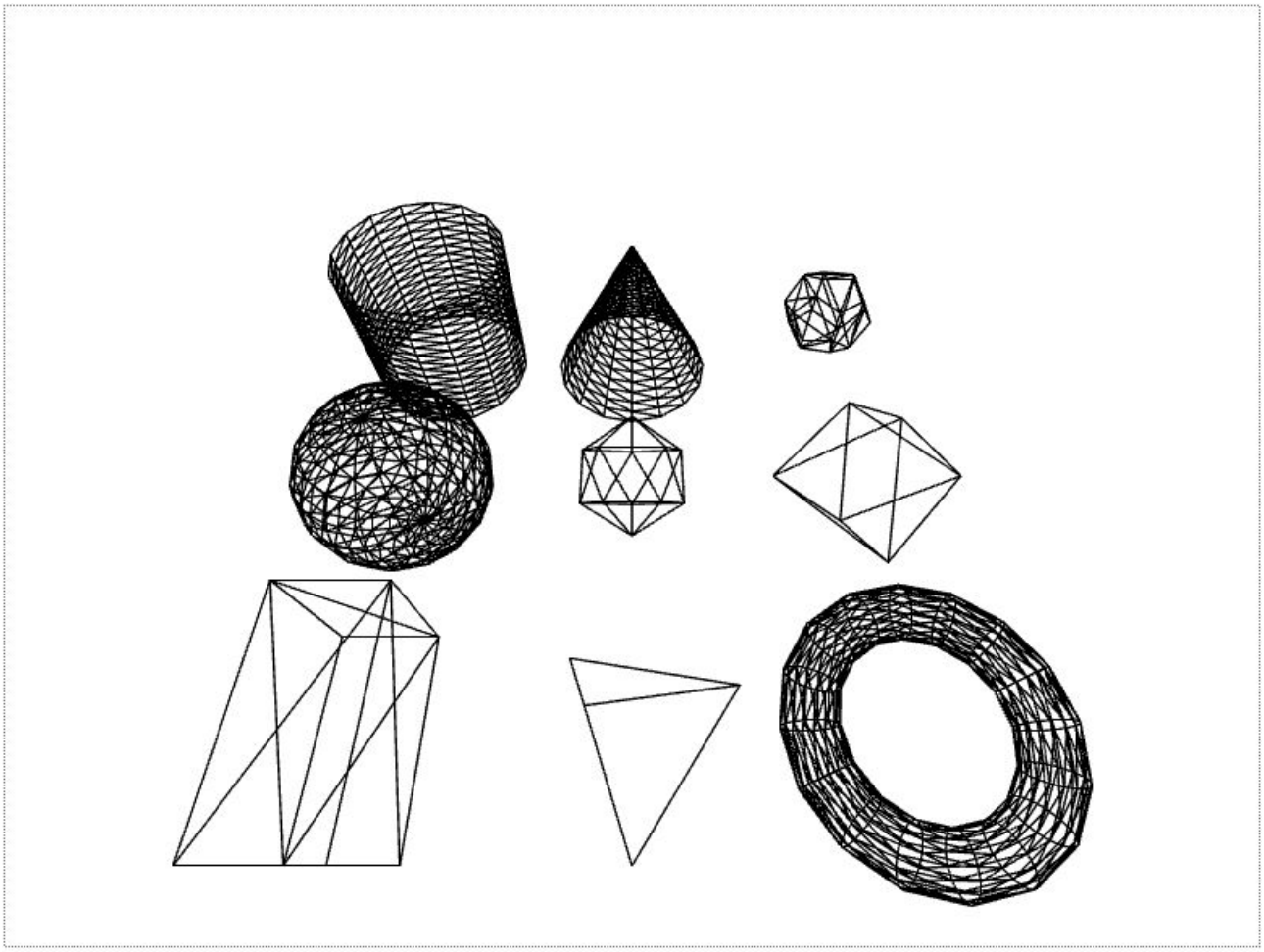
Cómo pueden notar en los ejemplos de objetos geométricos, la esfera y otros objetos, tienen caras formadas por cuadriláteros; y de los ejemplos de primitivas en WebGL pueden notar que WebGL no permite dibujar directamente cuadriláteros, sus primitivas son: puntos, líneas y triángulos. Así que parte del reto de la práctica es que comprendan cómo están construidos los objetos geométricos y modifiquen el código para generar dos triángulos por cada cuadrilátero que se construya.



Punto extra: agregar un:

```
<input type="checkbox" id="wire_ckbx"><label for=”wire_ckbx”>Wireframe</label>
```

y el código necesario para que, al activar el checkbox, los objetos geométricos se dibujen en modo aristas (wireframe); y al estar desactivado se dibuje en modo sólido.



☑ Wireframe

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 (22 de abril 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.