Ejercicio 1 – Introducción al desarrollo web

Este ejercicio tiene como objetivo que os empecéis a familiarizar con las tecnologías básicas del desarrollo web en su lado cliente: HTML, CSS y JavaScript. Para ello vamos a "jugar" con ejemplos de cada tecnología, siguiendo algunas de las partes del tutorial online https://www.w3schools.com/.

No hay que subir ninguno de los ficheros realizados en cada parte al Aula Virtual.

Puntos totales posibles del ejercicio: 0

Parte 1 - HTML

1. Lee la página de introducción a HTML. Comprueba que puedes modificar el ejemplo "*A Simple HTML Document*" en el editor online (botón "*Try it Yourself*").
2. Lee la página de editores HTML. Copia el contenido de la página web ejemplo a un fichero de tu disco duro local (por ejemplo "index.html") usando un editor de texto (Atom u otro que prefieras). Abre el resultado desde un navegador web instalado en tu ordenador (Chrome, Firefox, o el que prefieras).
3. Lee la página de elementos básicos de HTML. Averigua si tu editor de texto tiene funciones de autocompletado. Esta función se suele activar usando la combinación de teclado: Ctrl+Espacio en alguna posición del contenido de la página web que estás editando.

Parte 2 - CSS

4. Lee la página de uso de CSS desde HTML. Copia el contenido del ejemplo "External CSS" a ficheros locales, con extensiones ".html" y ".css". Comprueba si la función de autocompletado está disponible para el fichero CSS.
5. Lee la página de sintaxis CSS. Comprueba que entiendes todos los ejemplos, ejecutándolos mediante el editor online ("Try it yourself") o mediante una copia en local y usando un editor de textos.

Parte 3 - JavaScript

6. Lee la página de uso de JavaScript desde HTML. Comprueba que entiendes todos los ejemplos, ejecutándolos mediante el editor online ("Try it yourself") o mediante una copia en local y usando un editor de textos.
7. Lee la página sobre la localización del código JavaScript. Copia el contenido del ejemplo "External JavaScript" a ficheros locales, con extensiones ".html" y ".js". Comprueba si la función de autocompletado está disponible para el fichero JavaScript.
8. Lee la página de variables JavaScript. Comprueba que entiendes todos los ejemplos, ejecutándolos mediante el editor online ("Try it yourself") o mediante una copia en local y usando un editor de textos.
9. Lee la página de funciones JavaScript. Comprueba que entiendes todos los ejemplos, ejecutándolos mediante el editor online ("Try it yourself") o mediante una copia en local y usando un editor de textos.

Ejercicio 2 – Conceptos básicos en WebGL (1ª parte)

Este ejercicio tiene como objetivo que os empecéis a familiarizar con las tecnologías básicas para crear gráficos con WebGL, esto es, la API JavaScript de WebGL, así como la API GLSL ES para crear shaders.

No hay que subir ninguno de los ficheros realizados en cada parte al Aula Virtual.

Puntos totales posibles del ejercicio: 0

Parte 1 – Colorear un canvas

1. Copia el ejemplo “colorear un canvas” visto en clase en un fichero HTML local. Comprueba que eres capaz de visualizarlo en un navegador (Chrome o Firefox).
2. Abre la consola de depuración y observa que no aparecen errores.
3. Cambia el tamaño del canvas: ancho=1000px, alto=500px.
4. Haz que el color con el que se colorea el canvas sea rosa sólido en lugar de negro.

Parte 2 – Dibujar un punto

5. Copia el ejemplo “dibujar un punto” visto en clase en un fichero HTML local. Comprueba que eres capaz de visualizarlo en un navegador (Chrome o Firefox).
6. Mueve el punto 0.5px a la derecha y 0.3px arriba.
7. Cambia el tamaño del punto a 20px.
8. Haz que el color con el que se colorea el punto sea rosa en lugar de rojo.
9. Extrae el contenido del código JavaScript en un fichero externo (extensión .js) y enlázalo con en la página HTML. Comprueba que la aplicación sigue funcionando correctamente.

Ejercicio 3 – Dibujar puntos de color haciendo click en canvas

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 3 de la asignatura “Conceptos básicos en WebGL”.

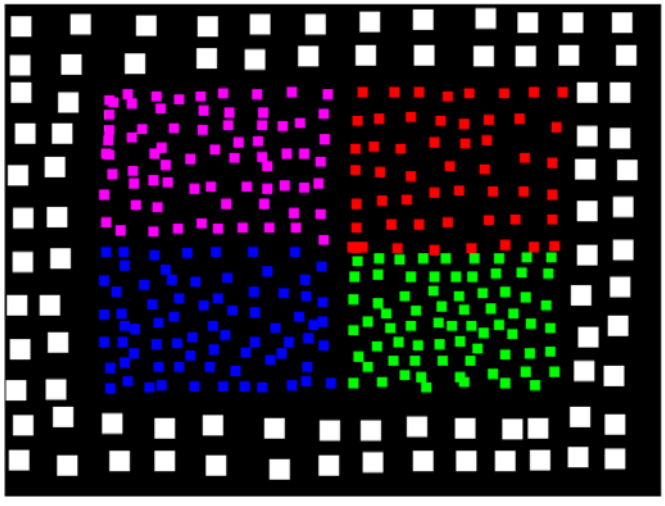
Como resultado de tu práctica deberás generar un **único fichero HTML** que deberás subir al Aula Virtual.

Puntos totales posibles del ejercicio: 10

Instrucciones

Partiendo de un canvas HTML, se pide realizar una aplicación WebGL que cumpla los siguientes requisitos:

- El color de fondo del canvas se pintará con WebGL en color **negro** (y permanecerá en ese color).
- Al iniciarse la aplicación el canvas no contendrá ningún gráfico.
- Al hacer click en el canvas, mediante JavaScript se localizará la posición de las coordenadas de ese click, pintando un **nuevo punto** en el canvas en dicha posición mediante WebGL
- El nuevo punto tendrá un **tamaño de 10 píxeles,** siempre y cuando se encuentre a una **distancia como máximo de 0.7 con respecto al centro**. Si el punto está a una **distancia mayor**, el tamaño será de **20 píxeles**.
- El color del punto estará determinado por la posición del click:
    - Color **blanco** para todo aquel punto con **distancia mayor a 0.7**.
    - Color **rojo** en el cuadrante **superior derecha** con distancia menor a 0.7.
    - Color **verde** en el cuadrante **inferior derecha** con distancia menor a 0.7.
    - Color **azul** en el cuadrante **inferior izquierda** con distancia menor a 0.7.
    - Color **rosa** en el cuadrante **superior izquierda** con distancia menor a 0.7.



Ayuda

Se proporciona el siguiente fragmento JavaScript, destinado a capturar el evento de click del ratón encima del canvas y transformar las coordenadas (x,y) de dicho click a coordenadas WebGL. Dado un canvas HTML5 identificado como "myCanvas":

```
// Get canvas object from the DOM
var canvas = document.getElementById("myCanvas");

// Init WebGL context
var gl = canvas.getContext("webgl");

// Register event handler
canvas.onmousedown = function(ev) {
    click(ev, gl, canvas);
};


function click(ev, gl, canvas) {
    // Coordinates of canvas origin
    var rect = ev.target.getBoundingClientRect();

    // relative x coordinate of click in canvas
    var clickX = ev.clientX - rect.left;

    // relative y coordinate of click in canvas
    var clickY = ev.clientY - rect.top;

    // WebGL coordinates (3D)
    var halfCanvasWidth = canvas.width / 2;
    var halfCanvasHeight = canvas.height / 2;
    var x = (clickX - halfCanvasWidth) / halfCanvasWidth;
    var y = (halfCanvasHeight - clickY) / halfCanvasHeight;
    var xyz = [x, y, 0];

    // ...
}
```

Ejercicio 3 – Dibujar puntos de color haciendo click en canvas

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 3 de la asignatura “Conceptos básicos en WebGL”.

Como resultado de tu práctica deberás generar un **único fichero HTML** que deberás subir al Aula Virtual.

Puntos totales posibles del ejercicio: 10

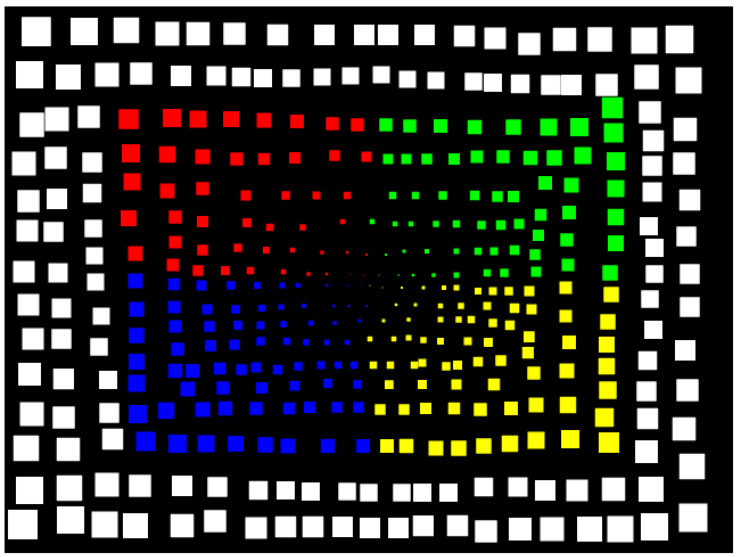
Instrucciones

Partiendo de un canvas HTML, se pide realizar una aplicación WebGL que cumpla los siguientes requisitos:

- El color de fondo del canvas se pintará con WebGL en color **negro** (y permanecerá en ese color).
- Al iniciarse la aplicación el canvas no contendrá ningún gráfico.
- Al hacer click en el canvas, mediante JavaScript se localizará la posición de las coordenadas de ese click, pintando un **nuevo punto** en el canvas en dicha posición mediante WebGL
- El nuevo punto tendrá un **tamaño que depende de su distancia al centro del canvas.** La cual se multiplicará por 20. Esta distancia se calcula como:

$$distancia = 20 * \sqrt{(punto\_x - centro\_x)^2 + (punto\_y - centro\_y)^2}$$

- El color del punto estará determinado por la posición del click:
  - Color **blanco** para todo aquel punto con **distancia mayor a 0.7**.
  - Color **verde** en el cuadrante **superior derecha** con distancia menor a 0.7.
  - Color **amarillo** en el cuadrante **inferior derecha** con distancia menor a 0.7.
  - Color **azul** en el cuadrante **inferior izquierda** con distancia menor a 0.7.
  - Color **rojo** en el cuadrante **superior izquierda** con distancia menor a 0.7.



Ayuda

Se proporciona el siguiente fragmento JavaScript, destinado a capturar el evento de click del ratón encima del canvas y transformar las coordenadas (x,y) de dicho click a coordenadas WebGL. Dado un canvas HTML5 identificado como "myCanvas":

```
// Get canvas object from the DOM
var canvas = document.getElementById("myCanvas");

// Init WebGL context
var gl = canvas.getContext("webgl");

// Register event handler
canvas.onmousedown = function(ev) {
    click(ev, gl, canvas);
};


function click(ev, gl, canvas) {
    // Coordinates of canvas origin
    var rect = ev.target.getBoundingClientRect();

    // relative x coordinate of click in canvas
    var clickX = ev.clientX - rect.left;

    // relative y coordinate of click in canvas
    var clickY = ev.clientY - rect.top;

    // WebGL coordinates (3D)
    var halfCanvasWidth = canvas.width / 2;
    var halfCanvasHeight = canvas.height / 2;
    var x = (clickX - halfCanvasWidth) / halfCanvasWidth;
    var y = (halfCanvasHeight - clickY) / halfCanvasHeight;
    var xyz = [x, y, 0];

    // ...
}
```

Ejercicio 3 – Dibujar puntos de color haciendo click en canvas

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 3 de la asignatura “Conceptos básicos en WebGL”.

Como resultado de tu práctica deberás generar un **único fichero HTML** que deberás subir al Aula Virtual.

Puntos totales posibles del ejercicio: 10

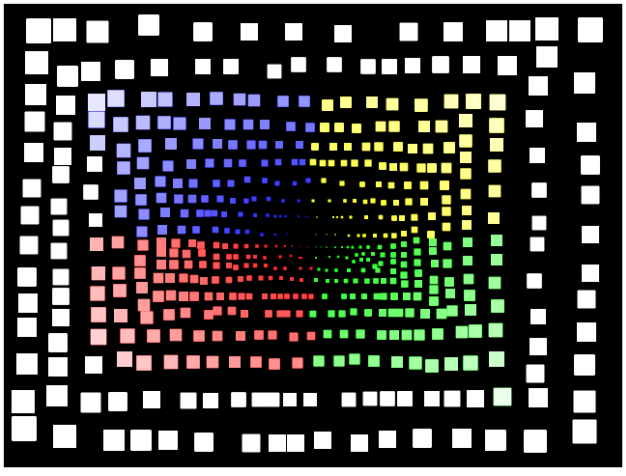
Instrucciones

Partiendo de un canvas HTML, se pide realizar una aplicación WebGL que cumpla los siguientes requisitos:

- El color de fondo del canvas se pintará con WebGL en color **negro** (y permanecerá en ese color).
- Al iniciarse la aplicación el canvas no contendrá ningún gráfico.
- Al hacer click en el canvas, mediante JavaScript se localizará la posición de las coordenadas de ese click, pintando un **nuevo punto** en el canvas en dicha posición mediante WebGL
- El nuevo punto tendrá un **tamaño que depende de su distancia al centro del canvas.** La cual se multiplicará por 20. Esta distancia se calcula como:

$$distancia = 20 * \sqrt{(punto\_x - centro\_x)^2 + (punto\_y - centro\_y)^2}$$

- El color del punto estará determinado por la posición del click:
  - Color **blanco** para todo aquel punto con **distancia mayor a 0.7**.
  - Color **amarillo** en el cuadrante **superior derecha** con distancia menor a 0.7.
  - Color **verde** en el cuadrante **inferior derecha** con distancia menor a 0.7.
  - Color **rojo** en el cuadrante **inferior izquierda** con distancia menor a 0.7.
  - Color **azul** en el cuadrante **superior izquierda** con distancia menor a 0.7.
- La transparencia del punto **depende de su distancia al centro del canvas.** Cuanto más cercano al borde (distancia 1), más transparente será (más próximo a 0).



Ayuda

Se proporciona el siguiente fragmento JavaScript, destinado a capturar el evento de click del ratón encima del canvas y transformar las coordenadas (x,y) de dicho click a coordenadas WebGL. Dado un canvas HTML5 identificado como "myCanvas":

```
// Get canvas object from the DOM
var canvas = document.getElementById("myCanvas");

// Init WebGL context
var gl = canvas.getContext("webgl");

// Register event handler
canvas.onmousedown = function(ev) {
    click(ev, gl, canvas);
};


function click(ev, gl, canvas) {
    // Coordinates of canvas origin
    var rect = ev.target.getBoundingClientRect();

    // relative x coordinate of click in canvas
    var clickX = ev.clientX - rect.left;

    // relative y coordinate of click in canvas
    var clickY = ev.clientY - rect.top;

    // WebGL coordinates (3D)
    var halfCanvasWidth = canvas.width / 2;
    var halfCanvasHeight = canvas.height / 2;
    var x = (clickX - halfCanvasWidth) / halfCanvasWidth;
    var y = (halfCanvasHeight - clickY) / halfCanvasHeight;
    var xyz = [x, y, 0];

    // ...
}
```

Ejercicio 3 – Dibujar triángulos de color haciendo click en canvas

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 3 de la asignatura “Conceptos básicos en WebGL”.

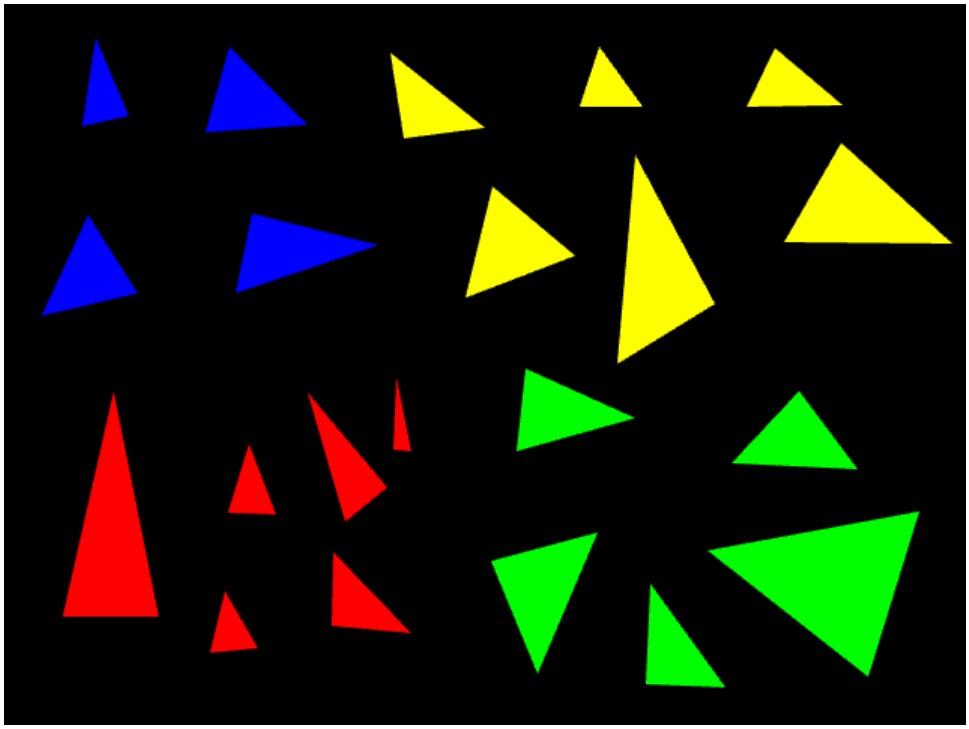
Como resultado de tu práctica deberás generar un **único fichero HTML** que deberás subir al Aula Virtual.

Puntos totales posibles del ejercicio: 10

Instrucciones

Partiendo de un canvas HTML, se pide realizar una aplicación WebGL que cumpla los siguientes requisitos:

- El color de fondo del canvas se pintará con WebGL en color **negro** (y permanecerá en ese color).
- Al iniciarse la aplicación el canvas no contendrá ningún gráfico.
- Al hacer click en el canvas, mediante JavaScript se localizará la posición de las coordenadas de ese click. Cuando haya al menos 3 puntos, se dibujará un **nuevo triángulo** en el canvas utilizando esos tres puntos marcados mediante WebGL.
- El color del triángulo estará determinado por la posición del centroide:
    - Color **amarillo** en el cuadrante **superior derecha**.
    - Color **verde** en el cuadrante **inferior derecha**.
    - Color **rojo** en el cuadrante **inferior izquierda**.
    - Color **azul** en el cuadrante **superior izquierda**.



Ayuda

Se proporciona el siguiente fragmento JavaScript, destinado a capturar el evento de click del ratón encima del canvas y transformar las coordenadas (x,y) de dicho click a coordenadas WebGL. Dado un canvas HTML5 identificado como "myCanvas":

```
// Get canvas object from the DOM
var canvas = document.getElementById("myCanvas");

// Init WebGL context
var gl = canvas.getContext("webgl");

// Register event handler
canvas.onmousedown = function(ev) {
    click(ev, gl, canvas);
};


function click(ev, gl, canvas) {
    // Coordinates of canvas origin
    var rect = ev.target.getBoundingClientRect();

    // relative x coordinate of click in canvas
    var clickX = ev.clientX - rect.left;

    // relative y coordinate of click in canvas
    var clickY = ev.clientY - rect.top;

    // WebGL coordinates (3D)
    var halfCanvasWidth = canvas.width / 2;
    var halfCanvasHeight = canvas.height / 2;
    var x = (clickX - halfCanvasWidth) / halfCanvasWidth;
    var y = (halfCanvasHeight - clickY) / halfCanvasHeight;
    var xyz = [x, y, 0];

    // ...
}
```

Dado un triángulo formado por los puntos ABC, las coordenadas del centroide O de un pueden calcularse como:

$$O_x = \frac{A_x + B_x + C_x}{3} \qquad O_y = \frac{A_y + B_y + C_y}{3}$$

Donde Ax y Ay son las coordenadas x e y del punto A, Bx y By del punto B y Cx y Cy del punto C.

Ejercicio 4 – Transformaciones con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 4 de la asignatura "Transformaciones con WebGL".

Como resultado de tu práctica deberás generar un **único fichero HTML** que deberás subir al Aula Virtual.

Puntos totales posibles del ejercicio: 10

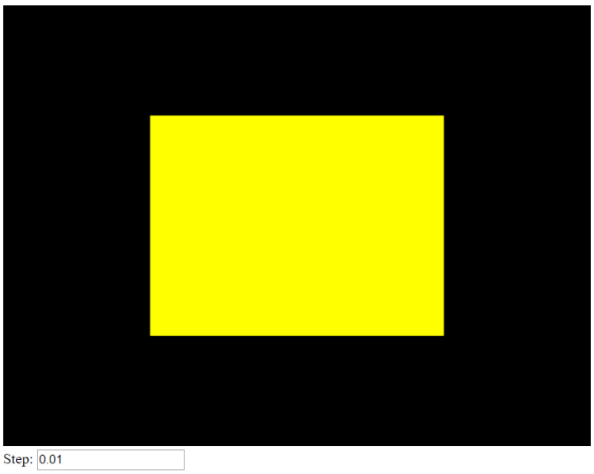
Instrucciones

Partiendo de un canvas HTML, se pide realizar una aplicación WebGL que cumpla los siguientes requisitos:

- El **color de fondo** del canvas se pintará con WebGL en color **negro** (y permanecerá en ese color).
- Al iniciarse la aplicación el canvas contendrá un **cuadrado amarillo centrado** en el canvas.
- Se capturan las **pulsaciones de teclado** en la página web, de modo que se realizarán las siguientes transformaciones al triángulo en función de la tecla pulsada:
    - **Flecha izquierda**: Traslación a la izquierda del eje X (el triángulo se mueve la izquierda)
    - **Flecha derecha**: Traslación a la derecha del eje X (el triángulo se mueve la derecha)
    - **Flecha arriba**: Traslación hacia arriba en el eje Y (el triángulo se mueve hacia arriba)
    - **Flecha abajo**: Traslación hacia abajo en el eje Y (el triángulo se mueve hacia abajo)
    - **Tecla RePag (PgUp)**: Aumento de escala (el triángulo se hace más grande)
    - **Tecla AvPag (PgDown)**: Contracción de escala (el triángulo se hace más pequeño)
    - **Tecla Inicio (Home)**: Giro negativo según el eje Z
    - **Tecla Fin (End)**: Giro positivo según el eje Z

Asegúrate de desactivar el bloqueo numérico si usas estas teclas

- El **valor** del cambio para cada una de las transformaciones será **configurable desde la interfaz** de usuario de la página web. En otras palabras, habrá un **campo de texto** que será leído desde JavaScript para averiguar el valor de cambio de la transformación. Por defecto este campo tendrá un valor de 0.01.



Ayuda

Suponiendo que el cuerpo de la pagina web es de la siguiente forma:

```
<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas>
    <br>
    Step: <input type="text" value="0.01" id="step">
</body>
```

Se puede usar el siguiente fragmento JavaScript para capturar la pulsación de teclado y evaluar las teclas requeridas:

```
document.onkeydown = function (ev) {
    var step = new Number(document.getElementById("step").value);
    switch (ev.keyCode) {
        case 37: // Left
            // ...
            break;
        case 39: // Right
            // ...
            break;
        case 38: // Up
            // ...
            break;
        case 40: // Down
            // ...
            break;
        case 33: // PageUp
            // ...
            break;
        case 34: // PageDown
            // ...
            break;
        case 36: // Home
            // ...
            break
        case 35: // End
            // ...
            break;
    }
    // ...
};
```

Ejercicio 4 – Transformaciones con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 4 de la asignatura "Transformaciones con WebGL".

Como resultado de tu práctica deberás generar un **único fichero HTML** que deberás subir al Aula Virtual.

Puntos totales posibles del ejercicio: 10

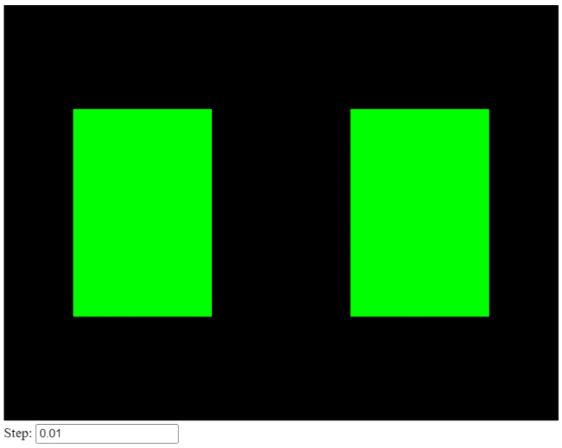
Instrucciones

Partiendo de un canvas HTML, se pide realizar una aplicación WebGL que cumpla los siguientes requisitos:

- El **color de fondo** del canvas se pintará con WebGL en color **negro** (y permanecerá en ese color).
- Al iniciarse la aplicación el canvas contendrá **dos rectángulos verdes centrados y separados** en el canvas.
- Se capturan las **pulsaciones de teclado** en la página web, de modo que se realizarán las siguientes transformaciones a los rectángulos en función de la tecla pulsada:
  - **Flecha izquierda**: Traslación a la izquierda del eje X (los rectángulos se mueven la izquierda)
  - **Flecha derecha**: Traslación a la derecha del eje X (los rectángulos se mueven la derecha)
  - **Flecha arriba**: Traslación hacia arriba en el eje Y (los rectángulos se mueven hacia arriba)
  - **Flecha abajo**: Traslación hacia abajo en el eje Y (los rectángulos se mueven hacia abajo)
  - **Tecla RePag (PgUp)**: Aumento de escala (los rectángulos se hacen más grande)
  - **Tecla AvPag (PgDown)**: Contracción de escala (los rectángulos se hacen más pequeño)
  - **Tecla Inicio (Home)**: Giro negativo según el eje Z
  - **Tecla Fin (End)**: Giro positivo según el eje Z
  - **Tecla Más (+)**: Acerca los cuadrados en el eje X
  - **Tecla Menos (-)**: Aleja los cuadrados en el eje X

Asegúrate de desactivar el bloqueo numérico si usas estas teclas

- El **valor** del cambio para cada una de las transformaciones será **configurable desde la interfaz** de usuario de la página web. En otras palabras, habrá un **campo de texto** que será leído desde JavaScript para averiguar el valor de cambio de la transformación. Por defecto este campo tendrá un valor de 0.01.



Ayuda

Suponiendo que el cuerpo de la pagina web es de la siguiente forma:

```
<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas>
    <br>
    Step: <input type="text" value="0.01" id="step">
</body>
```

Se puede usar el siguiente fragmento JavaScript para capturar la pulsación de teclado y evaluar las teclas requeridas:

```
document.onkeydown = function (ev) {
    var step = new Number(document.getElementById("step").value);
    switch (ev.keyCode) {
        case 37: // Left
            // ...
            break;
        case 39: // Right
            // ...
            break;
        case 38: // Up
            // ...
            break;
        case 40: // Down
            // ...
            break;
        case 33: // PageUp
            // ...
            break;
        case 34: // PageDown
            // ...
            break;
        case 36: // Home
            // ...
            break
        case 35: // End
            // ...
            break;
        case 187: // +
            // ...
            break;
        case 189: // -
            // ...
            break;
    }
    // ...
};
```

Ejercicio 4 – Transformaciones con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 4 de la asignatura "Transformaciones con WebGL".

Como resultado de tu práctica deberás generar un **único fichero HTML** que deberás subir al Aula Virtual.

Puntos totales posibles del ejercicio: 10

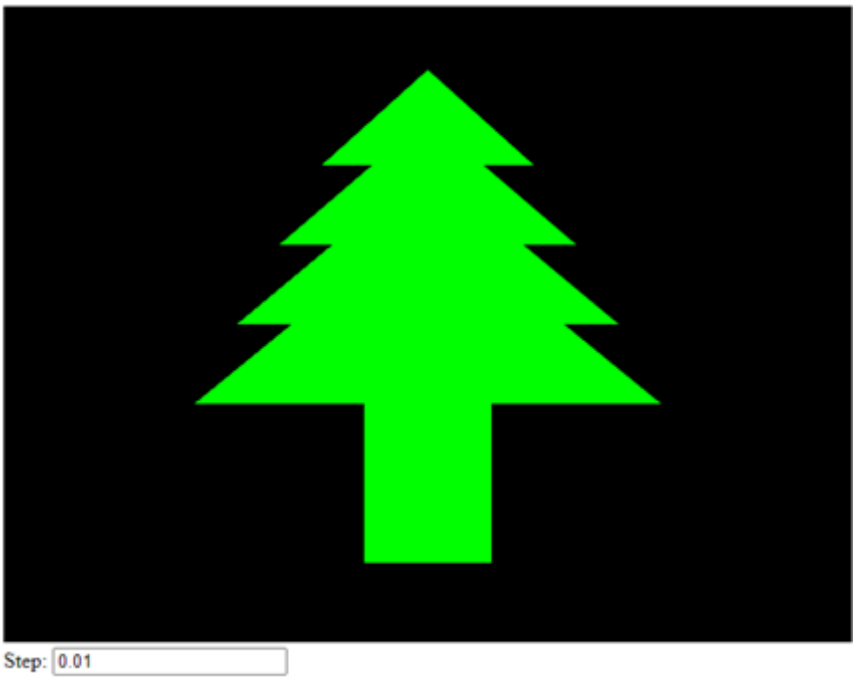
Instrucciones

Partiendo de un canvas HTML, se pide realizar una aplicación WebGL que cumpla los siguientes requisitos:

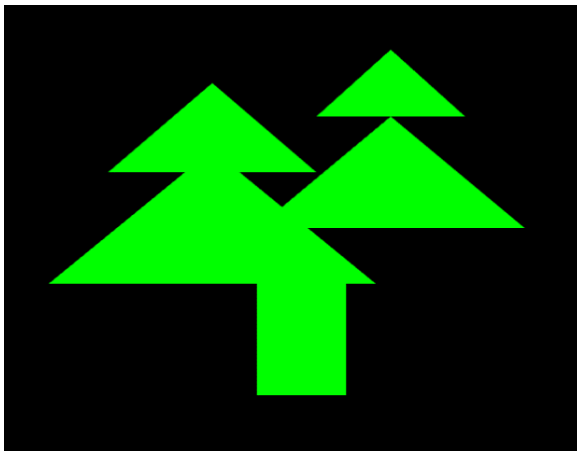
- El **color de fondo** del canvas se pintará con WebGL en color **negro** (y permanecerá en ese color).
- Al iniciarse la aplicación el canvas contendrá **un árbol formado por 4 niveles de hojas y la base de color verde** en el canvas (ver figura).
- Se capturan las **pulsaciones de teclado** en la página web, de modo que se realizarán las siguientes transformaciones a los rectángulos en función de la tecla pulsada:
  - **Flecha izquierda**: Traslación a la izquierda del eje X (el árbol se mueve a la izquierda)
  - **Flecha derecha**: Traslación a la derecha del eje X (el árbol se mueve a la derecha)
  - **Flecha arriba**: Traslación hacia arriba en el eje Y (el árbol se mueve hacia arriba)
  - **Flecha abajo**: Traslación hacia abajo en el eje Y (el árbol se mueve hacia abajo)
  - **Tecla RePag (PgUp)**: Aumento de escala (el árbol se hace más grande)
  - **Tecla AvPag (PgDown)**: Contracción de escala (el árbol se hace más pequeño)
  - **Tecla Inicio (Home)**: Giro negativo según el eje Z
  - **Tecla Fin (End)**: Giro positivo según el eje Z
  - **Tecla Más (+)**: Mueve los niveles impares del árbol a la derecha y los pares a la izquierda
  - **Tecla Menos (-)**: Mueve los niveles pares del árbol a la derecha y los impares a la izquierda

Asegúrate de desactivar el bloqueo numérico si usas estas teclas

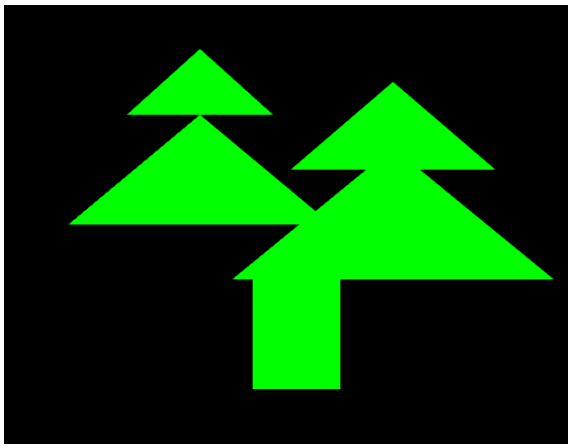
- El **valor** del cambio para cada una de las transformaciones será **configurable desde la interfaz** de usuario de la página web. En otras palabras, habrá un **campo de texto** que será leído desde JavaScript para averiguar el valor de cambio de la transformación. Por defecto este campo tendrá un valor de 0.01.



Pulsar +



Pulsar -



Ayuda

Suponiendo que el cuerpo de la página web es de la siguiente forma:

```
<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas>
    <br>
    Step: <input type="text" value="0.01" id="step">
</body>
```

Se puede usar el siguiente fragmento JavaScript para capturar la pulsación de teclado y evaluar las teclas requeridas:

```
document.onkeydown = function (ev) {
    var step = new Number(document.getElementById("step").value);
    switch (ev.keyCode) {
        case 37: // Left
            // ...
            break;
        case 39: // Right
            // ...
            break;
        case 38: // Up
            // ...
            break;
        case 40: // Down
            // ...
            break;
        case 33: // PageUp
            // ...
            break;
        case 34: // PageDown
            // ...
            break;
        case 36: // Home
            // ...
            break
        case 35: // End
            // ...
            break;
         case 187: // +
            // ...
            break;
         case 189: // -
            // ...
            break;
    }
    // ...
};
```

Ejercicio 4 – Transformaciones avanzadas con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL de forma avanzada, poniendo en práctica todos los conceptos estudiados en el tema 4 de la asignatura "Transformaciones con WebGL".

No hay que subir ninguno de los ficheros realizados en cada parte al Aula Virtual.

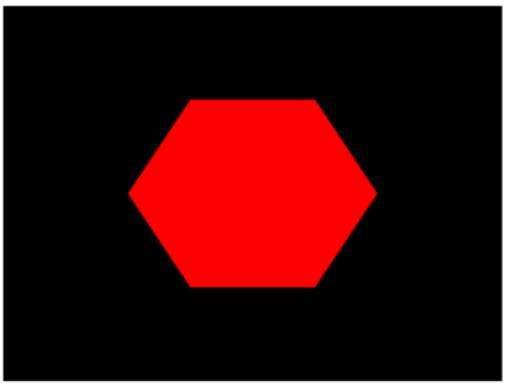
Puntos totales posibles del ejercicio: 0

Instrucciones

Partiendo de un canvas HTML, se pide realizar varias aplicaciones WebGL que cumplan los siguientes requisitos:

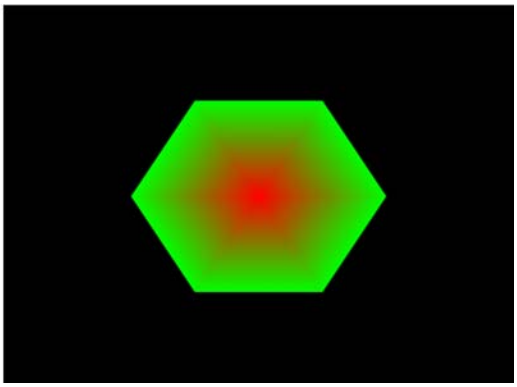
Apartado 1:

- El **color de fondo** del canvas se pintará con WebGL en color **negro**.
- Al iniciarse la aplicación el canvas contendrá un **hexágono rojo centrado** en el canvas **sin** utilizar **índices** ni la función **drawElements**.



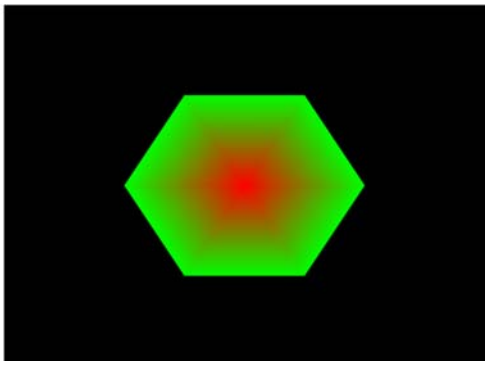
Apartado 2:

- Partiendo del **apartado 1**, se pide crear un hexágono cuyo color varíe, siendo el **vértice central rojo y el resto verde**.



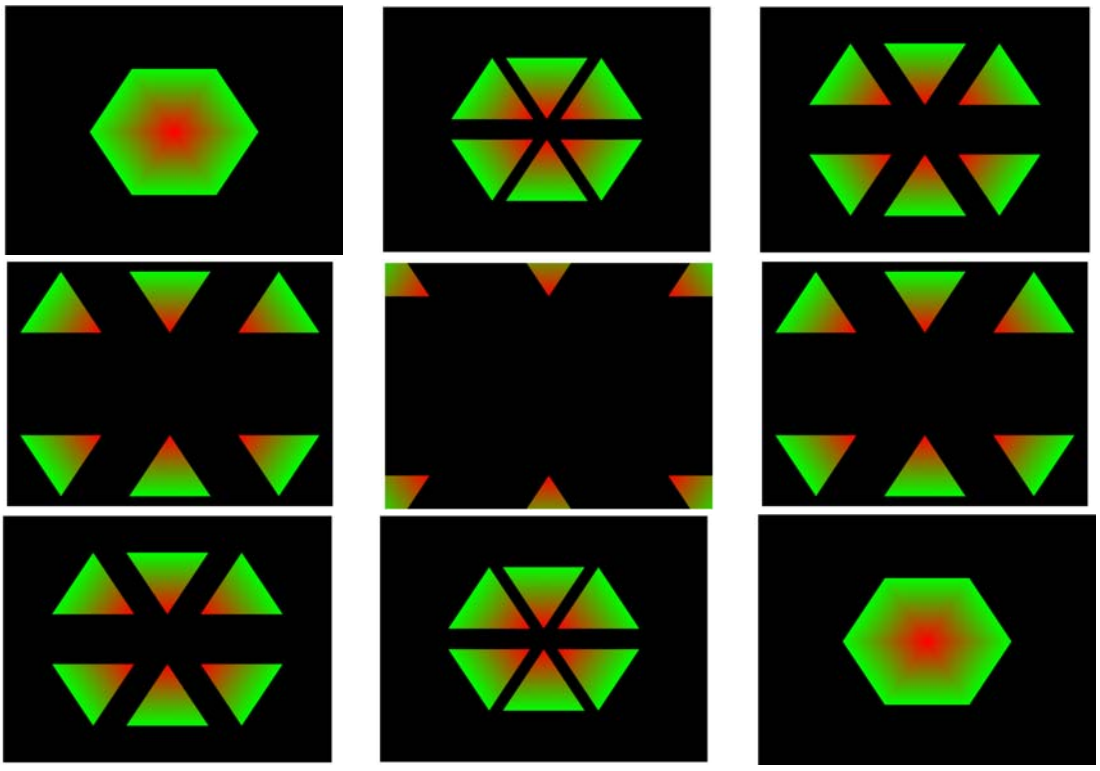
Apartado 3:

- Partiendo del **apartado 2**, crea el mismo hexágono, pero utilizando **índices** y la función **drawElements**.



Apartado 4:

- Partiendo del **apartado 3**, crea el mismo hexágono, pero haz que sus triángulos varíen en distintas direcciones, de manera que parezca que ha sufrido una explosión.



Ejercicio 5 – Proyecciones con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 5 de la asignatura “Proyecciones con WebGL”.

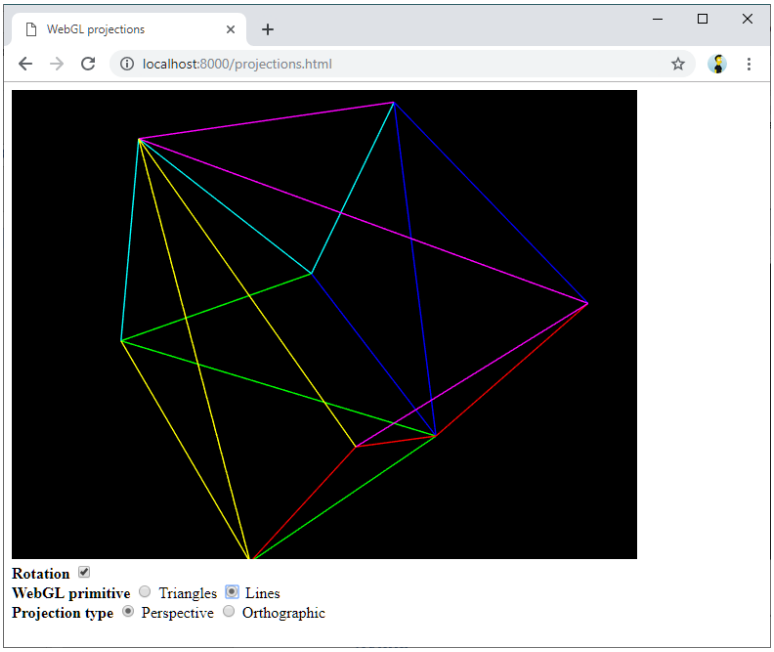
Como resultado de tu práctica deberás generar un **único fichero HTML** que deberás subir al Aula Virtual.

Puntos totales posibles del ejercicio: 10

Instrucciones

Partiendo del ejemplo visto en clase “proyección en perspectiva”, se pide hacer las siguientes modificaciones:

1. La rotación del cubo se podrá desactivar desde la interfaz de usuario mediante una casilla de verificación (*checkbox*).
2. La primitiva WebGL usada para dibujar la escena se podrá elegir desde la interfaz de usuario mediante botones de opción (*radio buttons*). Las primitivas serán triángulos (`gl.TRIANGLES`, opción por defecto) y líneas (`gl.LINES`).
3. El tipo de proyección usado para dibujar la escena se podría elegir desde la interfaz de usuario mediante botones de opción (*radio buttons*). Las opciones son 2: proyección en perspectiva (opción por defecto) y ortogonal.
4. Se deberá incluir un manejador de evento que escuche la rueda del ratón (`wheel`) en toda la página web de modo que al girar la rueda hacia delante se incremente en una unidad la coordenada z de la posición inicial de la cámara (implementada con la función `mat4.lookAt()` del ejemplo original). Cuando la rueda gire en el sentido inverso, la coordenada z de la posición de la cámara se decrementará en 1 unidad.



Ayuda

Puedes incluir los controles necesarios en la interfaz de usuario como sigue:

```
<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas><br>
    <b>Rotation</b>
    <input type="checkbox" name="rotation" checked><br>
    <b>WebGL primitive</b>
    <input type="radio" name="primitive" value="triangles" checked> Triangles
    <input type="radio" name="primitive" value="lines"> Lines<br>
    <b>Projection type</b>
    <input type="radio" name="projection" value="perspective" checked> Perspective
    <input type="radio" name="projection" value="orthographic"> Orthographic<br>
</body>
```

Para leer los valores de los diferentes campos (checkbox, radio) puedes usar las siguientes sentencias en JavaScript:

```
var rotationChecked = document.querySelector('input[name="rotation"]:checked');
var primitiveValue = document.querySelector('input[name="primitive"]:checked').value;
var projectionValue = document.querySelector('input[name="projection"]:checked').value;
```

Los valores recomendados para la matriz de proyección ortogonal es el siguiente (la variable `canvas` identifica el canvas HTML5):

```
var ratio = canvas.width / canvas.height;
var pMatrix = mat4.ortho(mat4.create(), -ratio, ratio, 1.0, -1.0, 5.0, -1.0);
```

El manejador de eventos para controlar la rueda del ratón se puede implementar usando el siguiente fragmento de código. Ten en cuenta que la variable `z` se va a utilizar como coordenadas z de la posición de la cámara (`mat4.lookAt()`) únicamente cuando se utiliza la vista en perspectiva. El valor por defecto de esta coordenada en el ejemplo original es `-3`:

```
// Event listener for mouse wheel
document.addEventListener('wheel', function (event) {
    z = event.wheelDelta > 0 ? z + 1 : z - 1;
});
```

Ejercicio 5 – Proyecciones con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 5 de la asignatura “Proyecciones con WebGL”.

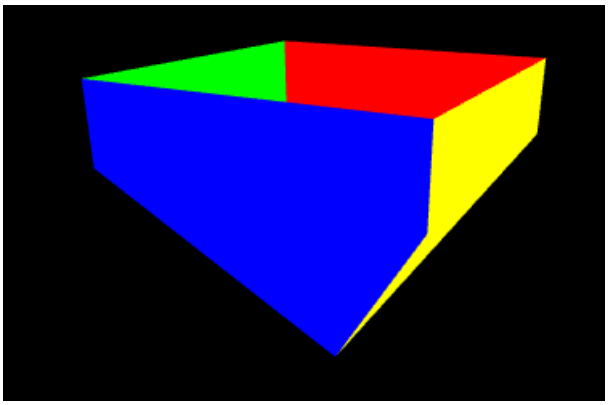
Como resultado de tu práctica deberás generar un **único fichero HTML** que deberás subir al Aula Virtual.

Puntos totales posibles del ejercicio: 10

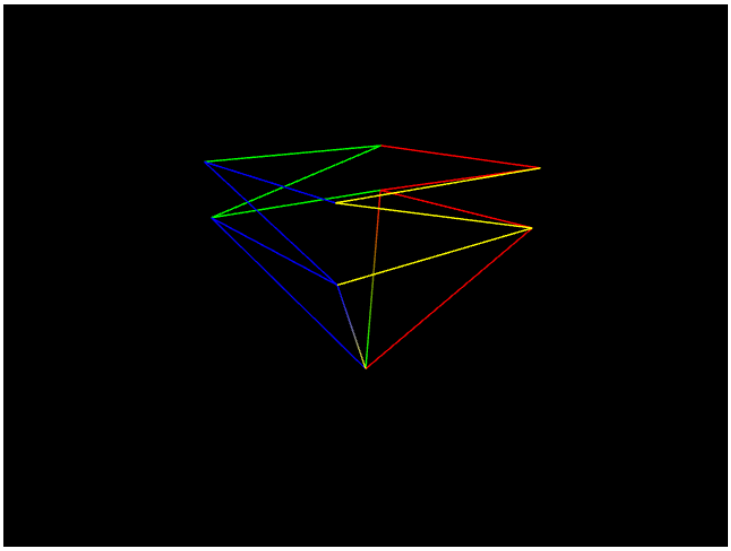
Instrucciones

Partiendo del ejemplo visto en clase “proyección en perspectiva”, se pide hacer las siguientes modificaciones:

1. En lugar de un cubo, se dibujará la siguiente figura, la cual consta de una pirámide invertida de 4 lados, un rectángulo por cada lateral de la base, y la base abierta. Además esta figura estará rotando constantemente solo sobre su eje Y.



2. La rotación de la figura se podrá desactivar desde la interfaz de usuario mediante una casilla de verificación (*checkbox*).
3. La primitiva WebGL usada para dibujar la escena se podrá elegir desde la interfaz de usuario mediante botones de opción (*radio buttons*). Las primitivas serán triángulos (`gl.TRIANGLES`, opción por defecto) y líneas (`gl.LINES`).
4. El tipo de proyección usado para dibujar la escena se podría elegir desde la interfaz de usuario mediante botones de opción (*radio buttons*). Las opciones son 2: proyección en perspectiva (opción por defecto) y ortogonal.
5. Se deberá incluir un manejador de evento que escuche la rueda del ratón (`wheel`) en toda la página web de modo que al girar la rueda hacia delante se incremente en una unidad la coordenada z de la posición inicial de la cámara (implementada con la función `mat4.lookAt()` del ejemplo original). Cuando la rueda gire en el sentido inverso, la coordenada z de la posición de la cámara se decrementará en 1 unidad.



Rotation ☑
WebGL primitive ○ Triangles ◉ Lines
Projection type ◉ Perspective ○ Orthographic

Ayuda

Puedes incluir los controles necesarios en la interfaz de usuario como sigue:

```
<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas><br>
    <b>Rotation</b>
    <input type="checkbox" name="rotation" checked><br>
    <b>WebGL primitive</b>
    <input type="radio" name="primitive" value="triangles" checked> Triangles
    <input type="radio" name="primitive" value="lines"> Lines<br>
    <b>Projection type</b>
    <input type="radio" name="projection" value="perspective" checked> Perspective
    <input type="radio" name="projection" value="orthographic"> Orthographic<br>
</body>
```

Para leer los valores de los diferentes campos (checkbox, radio) puedes usar las siguientes sentencias en JavaScript:

```
var rotationChecked = document.querySelector('input[name="rotation"]:checked');
var primitiveValue = document.querySelector('input[name="primitive"]:checked').value;
var projectionValue = document.querySelector('input[name="projection"]:checked').value;
```

Los valores recomendados para la matriz de proyección ortogonal es el siguiente (la variable `canvas` identifica el canvas HTML5):

```
var ratio = canvas.width / canvas.height;
var pMatrix = mat4.ortho(mat4.create(), -ratio, ratio, -1.0, 1.0, -1.0, 1.0);
```

El manejador de eventos para controlar la rueda del ratón se puede implementar usando el siguiente fragmento de código. Ten en cuenta que la variable `z` se va a utilizar como coordenadas z de la posición de la cámara (`mat4.lookAt()`) únicamente cuando se utiliza la vista en perspectiva. El valor por defecto de esta coordenada en el ejemplo original es `-3`:

```
// Event listener for mouse wheel
document.addEventListener('wheel', function (event) {
    z = event.wheelDelta > 0 ? z + 1 : z - 1;
});
```

Ejercicio 5 – Proyecciones con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 5 de la asignatura “Proyecciones con WebGL”.

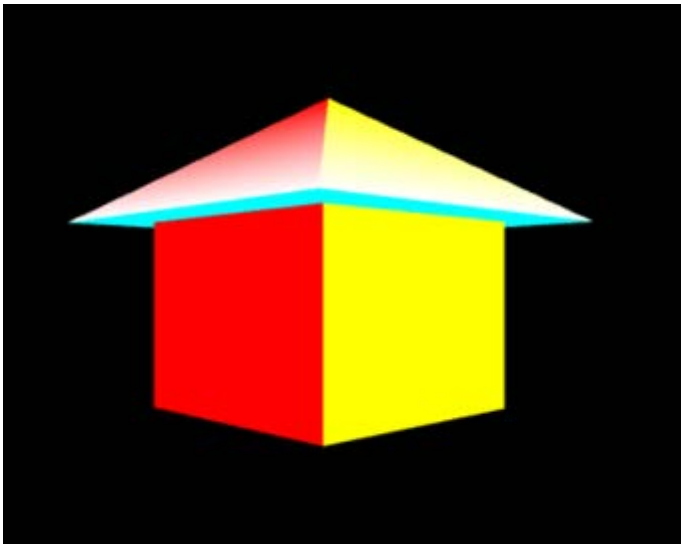
Como resultado de tu práctica deberás generar un **único fichero HTML** que deberás subir al Aula Virtual.

Puntos totales posibles del ejercicio: 10

Instrucciones

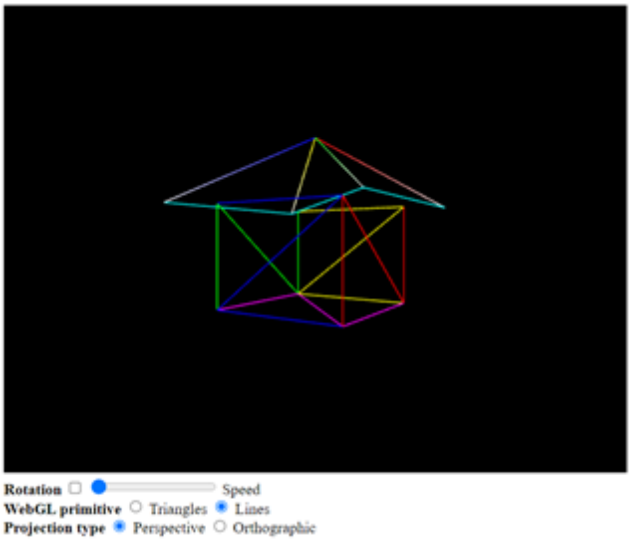
Partiendo del ejemplo visto en clase “proyección en perspectiva”, se pide hacer las siguientes modificaciones:

1. En lugar de un cubo, se dibujará la siguiente figura, la cual consta de una pirámide de 4 lados, y un cubo. Los colores pueden variar, pero hay que tener en cuenta que el degradado de la pirámide tiene que ser del color elegido para esa cara del cubo hacia el blanco. Además, esta figura estará rotando constantemente solo sobre su eje Y.



2. La rotación de la figura se podrá desactivar desde la interfaz de usuario mediante una casilla de verificación (*checkbox*).
3. La velocidad de rotación inicial (incremento de 0.01) se verá modificada de 1 a 3 a través de un botón de rango (*range*).
4. La primitiva WebGL usada para dibujar la escena se podrá elegir desde la interfaz de usuario mediante botones de opción (*radio buttons*). Las primitivas serán triángulos (`gl.TRIANGLES`, opción por defecto) y líneas (`gl.LINES`).
5. El tipo de proyección usado para dibujar la escena se podría elegir desde la interfaz de usuario mediante botones de opción (*radio buttons*). Las opciones son 2: proyección en perspectiva (opción por defecto) y ortogonal.

6. Se deberá incluir un manejador de evento que escuche la rueda del ratón (`wheel`) en toda la página web de modo que al girar la rueda hacia delante se decremente en una unidad la coordenada z de la posición inicial de la cámara (implementada con la función `mat4.lookAt()` del ejemplo original). Cuando la rueda gire en el sentido inverso, la coordenada z de la posición de la cámara se incrementará en 1 unidad.



Ayuda

Puedes incluir los controles necesarios en la interfaz de usuario como sigue:

```
<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas><br>
    <b>Rotation</b>
    <input type="checkbox" name="rotation" checked>
    <input type="range" id="speed" min="1" max="3" value="1" step="1"> Speed<br>
    <b>WebGL primitive</b>
    <input type="radio" name="primitive" value="triangles" checked> Triangles
    <input type="radio" name="primitive" value="lines"> Lines<br>
    <b>Projection type</b>
    <input type="radio" name="projection" value="perspective" checked> Perspective
    <input type="radio" name="projection" value="orthographic"> Orthographic<br>
</body>
```

Para leer los valores de los diferentes campos (checkbox, radio, range) puedes usar las siguientes sentencias en JavaScript:

```
var rotationChecked = document.querySelector('input[name="rotation"]:checked');
var primitiveValue = document.querySelector('input[name="primitive"]:checked').value;
var projectionValue = document.querySelector('input[name="projection"]:checked').value;
var speed = document.getElementById("speed").value;
```

El manejador de eventos para controlar la rueda del ratón se puede implementar usando el siguiente fragmento de código. Ten en cuenta que la variable `z` se va a utilizar como coordenadas z de la posición de la cámara (`mat4.lookAt()`) únicamente cuando se utiliza la vista en perspectiva. El valor por defecto de esta coordenada en el ejemplo original es `3`:

```
// Event listener for mouse wheel
document.addEventListener('wheel', function (event) {
    z = event.wheelDelta > 0 ? z - 1 : z + 1;
});
```

Ejercicio 5 – Proyecciones con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 5 de la asignatura “Proyecciones con WebGL”.

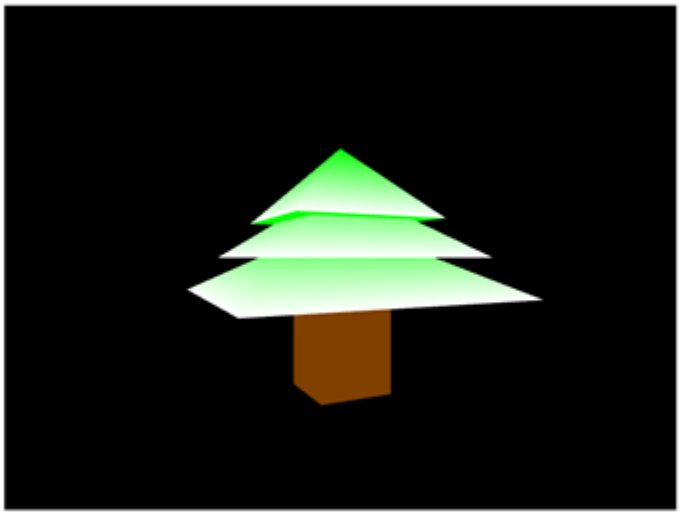
Como resultado de tu práctica deberás generar un **único fichero HTML** que deberás subir al Aula Virtual.

Puntos totales posibles del ejercicio: 10

Instrucciones

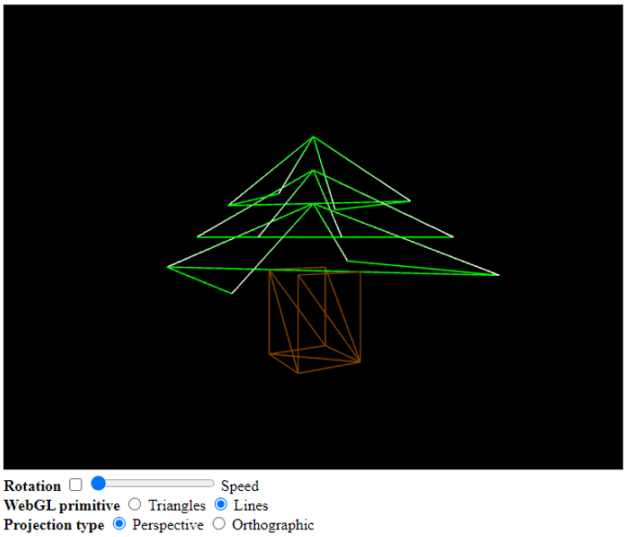
Partiendo del ejemplo visto en clase “proyección en perspectiva”, se pide hacer las siguientes modificaciones:

1. En lugar de un cubo, se dibujará un **árbol** similar a la de la práctica anterior, pero en 3D. Esta figura consta de **3 pirámides de 4 lados, y un cubo**. Al tratarse de un árbol, los **colores** serán **marrón para el tronco** (formado por el cubo), y **verde para las hojas** (formadas por las pirámides). En este caso las **pirámides** tendrán un **color degradado** del verde en el vértice superior, al blanco en los vértices inferiores. Por último, esta **figura** estará **rotando** constantemente solo sobre su **eje Y**.



2. La **rotación** de la figura se podrá desactivar desde la interfaz de usuario mediante una casilla de verificación (*checkbox*).
3. La **velocidad de rotación** inicial se verá modificada de 1 a 3 (con incremento de 1) a través de un botón de rango (*range*).
4. La **primitiva WebGL** usada para dibujar la escena se podrá elegir desde la interfaz de usuario mediante botones de opción (*radio buttons*). Las primitivas serán triángulos (`gl.TRIANGLES`, opción por defecto) o líneas (`gl.LINES`).
5. El **tipo de proyección** usado para dibujar la escena se podría elegir desde la interfaz de usuario mediante botones de opción (*radio buttons*). Las opciones son 2: proyección en **perspectiva** (opción por defecto) y **ortográfica**.

6. Se deberá incluir un manejador de evento que escuche la **rueda del ratón** (`wheel`) en toda la página web de modo que al girar la rueda hacia delante se decremente en una unidad la **coordenada z de la posición inicial de la cámara** (implementada con la función `mat4.lookAt()` del ejemplo original). Cuando la rueda gire en el sentido inverso, la coordenada z de la posición de la cámara se incrementará en 1 unidad.



Ayuda

Puedes incluir los controles necesarios en la interfaz de usuario como sigue:

```
<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas><br>
    <b>Rotation</b>
    <input type="checkbox" name="rotation" checked>
    <input type="range" id="speed" min="1" max="3" value="1" step="1"> Speed<br>
    <b>WebGL primitive</b>
    <input type="radio" name="primitive" value="triangles" checked> Triangles
    <input type="radio" name="primitive" value="lines"> Lines<br>
    <b>Projection type</b>
    <input type="radio" name="projection" value="perspective" checked> Perspective
    <input type="radio" name="projection" value="orthographic"> Orthographic<br>
</body>
```

Para leer los valores de los diferentes campos (checkbox, radio, range) puedes usar las siguientes sentencias en JavaScript:

```
var rotationChecked = document.querySelector('input[name="rotation"]:checked');
var primitiveValue = document.querySelector('input[name="primitive"]:checked').value;
var projectionValue = document.querySelector('input[name="projection"]:checked').value;
var speed = document.getElementById("speed").value;
```

El manejador de eventos para controlar la rueda del ratón se puede implementar usando el siguiente fragmento de código. Ten en cuenta que la variable `z` se va a utilizar como coordenadas z de la posición de la cámara (`mat4.lookAt()`) **únicamente** cuando se utiliza la **vista en perspectiva**. El valor por defecto de esta coordenada en el ejemplo original es `3`:

```
// Event listener for mouse wheel
document.addEventListener('wheel', function (event) {
    z = event.wheelDelta > 0 ? z - 1 : z + 1;
});
```

Ejercicio 5 – Proyecciones avanzadas con WebGL

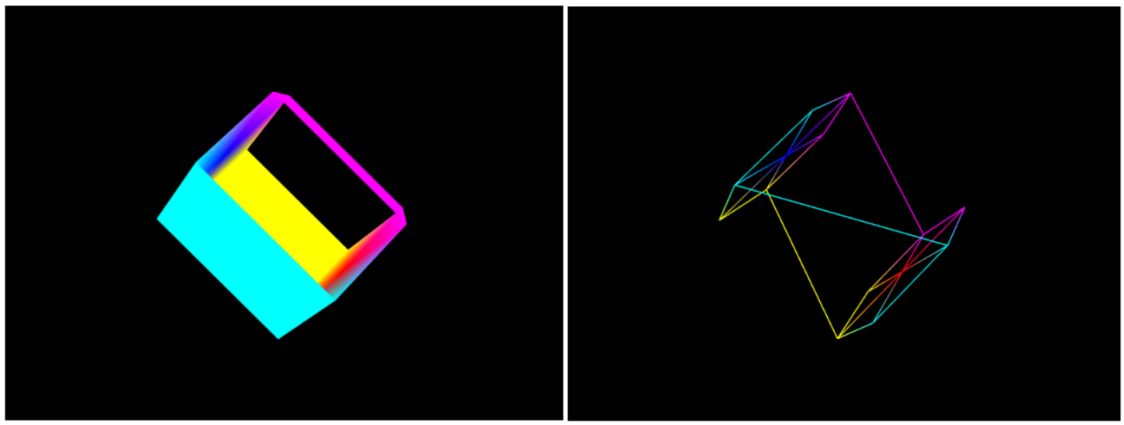
Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 5 de la asignatura “Proyecciones con WebGL”.

No hay que subir ninguno de los ficheros realizados en cada parte al Aula Virtual.

Puntos totales posibles del ejercicio: 0

Instrucciones

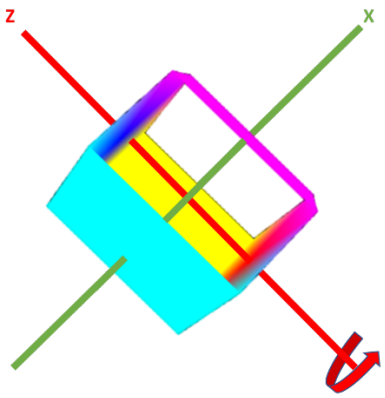
Partiendo de la práctica 5, se pide cambiar la figura por la siguiente:



Esta figura está formada por:

1. Dos hexágonos
2. Tres rectángulos que conectan los vértices de los hexágonos de forma alterna.

La figura debe estar girada en los ejes de tal forma que la rotación se realice sobre el eje Z:



Para ello, suponiendo que los hexágonos sean creados sobre el plano z = 0.5 y el z = -0.5, habrá que rotar 90º el eje X, seguido de una rotación de 45º sobre el eje Y. Recordad que las rotaciones sobre los ejes utilizando las funciones de glMatrix, se realizan en radianes.

Ejercicio 6 – Texturas con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 6 de la asignatura “Texturas con WebGL”.

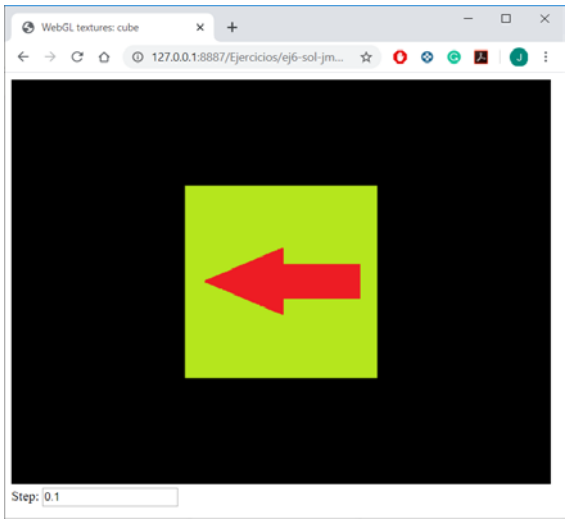
Como resultado de tu práctica deberás generar un **único fichero comprimido .zip** que deberás subir al Aula Virtual, que contendrá **un fichero HTML** y las **dos imágenes** de textura.

Puntos totales posibles del ejercicio: 10

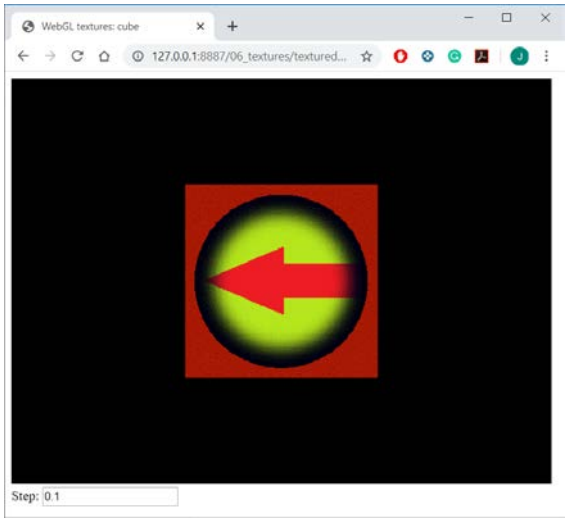
Instrucciones

Partiendo del ejemplo visto en clase “cubo con texturas”, se pide hacer las siguientes modificaciones:

1. **Cambia la textura** y añade la imagen **“arrow.jpg”**, inicialmente la **flecha** tiene que apuntar hacia la **izquierda**:



2. Añade la **segunda imagen “circles.gif”** a la textura al cubo con la textura de la flecha del apartado anterior:

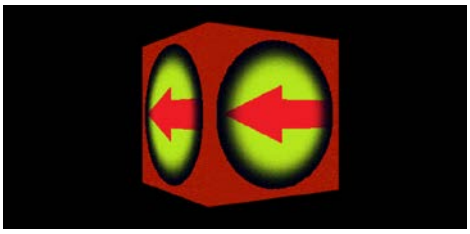


3. Se capturan las pulsaciones de teclado en la página web, de modo que se realizarán las siguientes transformaciones al cubo y a la cámara en función de la tecla pulsada:
    - **Flecha izquierda**: Rotación hacia la izquierda del cubo.
    - **Flecha derecha**: Rotación hacia la derecha del cubo.
    - **Flecha arriba**: Rotación hacia arriba del cubo.
    - **Flecha abajo**: Rotación hacia abajo del cubo.
    - **Tecla RePag (PgUp)**: Elevar la cámara.
    - **Tecla AvPag (PgDown)**: Descender la cámara.

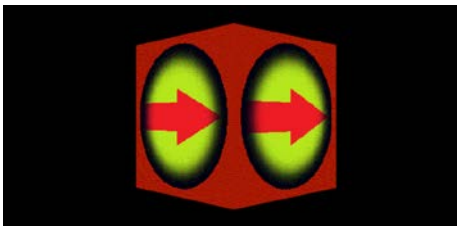
4. Al igual que en el ejercicio 4, el **valor**, que será la velocidad de giro y el paso en los desplazamientos de la cámara en el eje vertical, será **configurable desde la interfaz** de usuario de la página web. En otras palabras, habrá un campo de texto que será leído desde JavaScript para averiguar el valor de cambio de la transformación. Por defecto este campo tendrá un valor de **0.1**.

5. Como en el ejercicio 5, se deberá incluir un manejador de evento que escuche la **rueda del ratón** (`wheel`) de manera que el giro de la rueda varíe la distancia de la cámara con respecto al cubo e incremente/decremente esta distancia en 1 unidad.

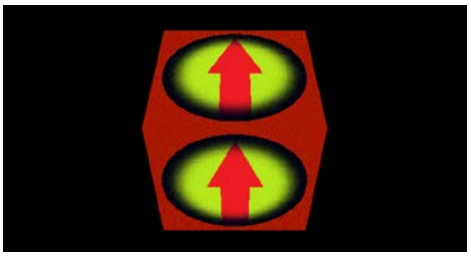
6. **Rota la textura** con cada tecla, de manera que la flecha indique el sentido de giro del cubo:



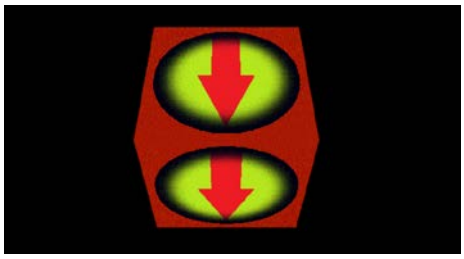
izquierda



derecha



arriba



abajo

Ayuda

Los movimientos de la cámara están asociados con la función `mat4.lookAt()`.

En las rotaciones, reinicia la rotación en el eje que no se mueve, es decir, si se realiza una rotación en el eje X, asigna una rotación 0 en el eje Y, y viceversa.

Ejercicio 6 – Texturas con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 6 de la asignatura “Texturas con WebGL”.

Como resultado de tu práctica deberás generar un **único fichero comprimido .zip** que deberás subir al Aula Virtual, que contendrá **un fichero HTML** y las **dos imágenes** de textura.

Puntos totales posibles del ejercicio: 10

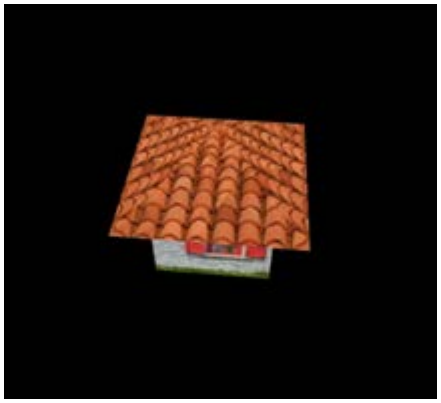
Instrucciones

Partiendo del ejercicio anterior, se pide hacer las siguientes modificaciones:

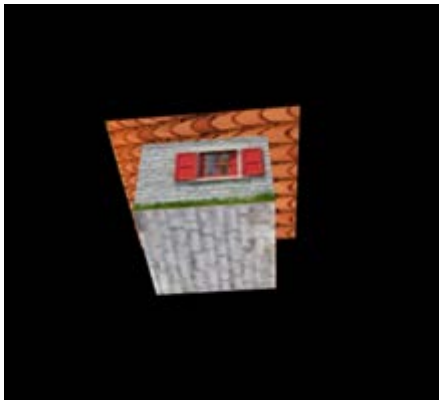
1. **Añade textura** a partir de la imagen **“house.jpg”**, según corresponda:
   A la **cara frontal** del cubo, le corresponde la parte que tiene la **puerta**, mientras que al resto de caras (**izquierda, derecha y trasera**) le corresponde la textura que contiene la **ventana**. Para la **pirámide** se pondrá la textura de las **tejas**, teniendo en cuenta la zona elegida, de manera que el triángulo sea el correcto (ver imagen). Para la **parte inferior**, se pide seleccionar de la textura una parte que corresponda solo a **piedra**, y no incluya ningún otro elemento.

Cara frontal

Caras laterales

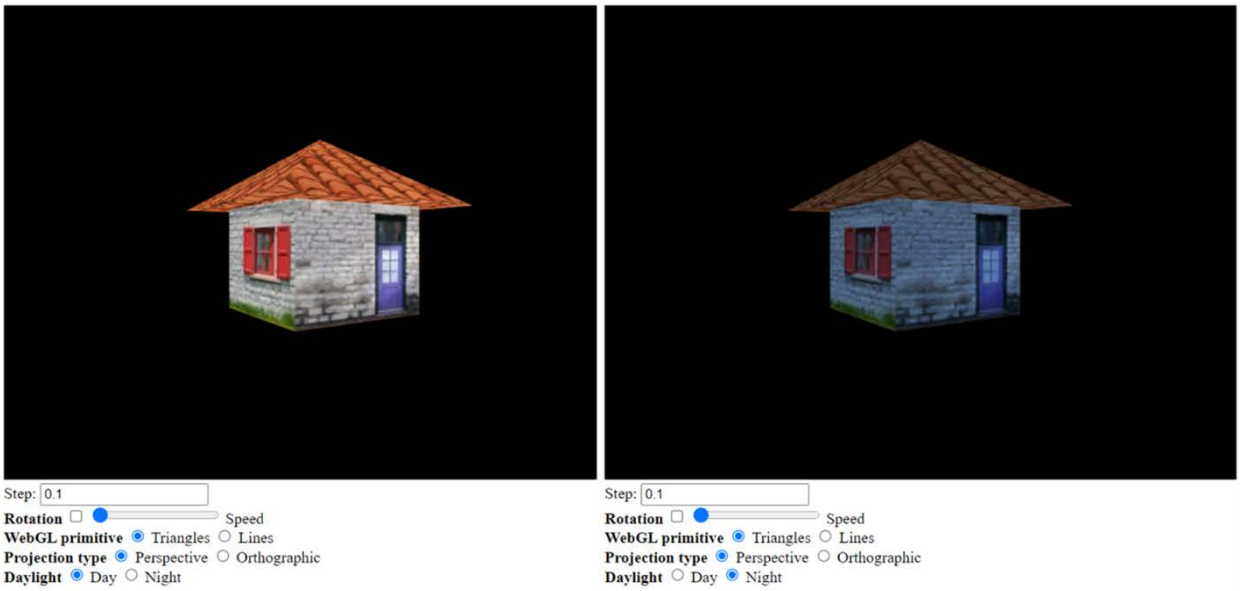


Pirámide



Cara inferior

2. Añade un **radio button** que permita elegir entre dos opciones: day / night, siendo day la opción elegida por defecto. Cuando el usuario elija la opción dark, se aplicará la **segunda imagen "dark.gif"** a la textura de la casa con la textura del apartado anterior. Cuando vuelva a elegir day, se dejará de aplicar esta segunda imagen y la textura será la original de la casa añadida en el apartado anterior.



3. Se capturan las pulsaciones de teclado en la página web, de modo que se realizarán las siguientes transformaciones sobre la cámara en función de la tecla pulsada y del valor configurable desde la interfaz de usuario:
    - **Tecla Flecha arriba:** Elevar la cámara.
    - **Tecla Flecha abajo:** Descender la cámara.

Ayuda

Los movimientos de la cámara están asociados con la función `mat4.lookAt()`.

Puedes incluir los nuevos controles necesarios en la interfaz de usuario como sigue:

```
<body onload="init()">
    […]
    <b>Daylight</b>
    <input type="radio" name="daylight" value="day" checked> Day
    <input type="radio" name="daylight" value="night"> Night<br>
</body>
```

Ejercicio 6 – Texturas con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 6 de la asignatura “Texturas con WebGL”.

Como resultado de tu práctica deberás generar un **único fichero comprimido .zip** que deberás subir al Aula Virtual, que contendrá **un fichero HTML** y las **dos imágenes** de textura.

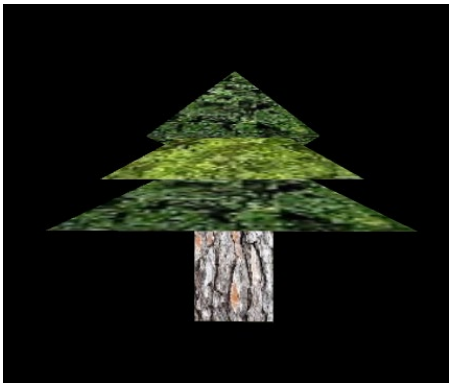
Puntos totales posibles del ejercicio: 10

Instrucciones

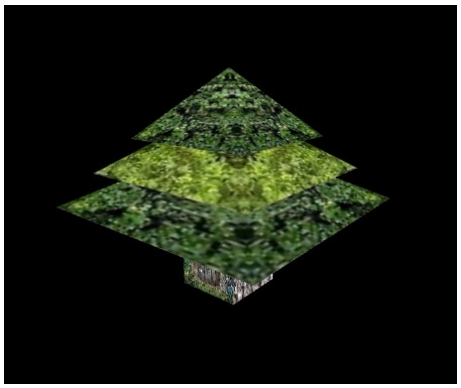
Partiendo del ejercicio anterior, se pide hacer las siguientes modificaciones:

1. **Añade textura** a partir de la imagen **“tree.jpg”**, según corresponda:
   A la **cara frontal** del cubo, le corresponde la parte que tiene la **puerta**, mientras que al resto de caras (**izquierda, derecha, trasera e inferior**) le corresponde la textura que contiene la **corteza de árbol**. Para las hojas, en las **pirámides superior e inferior** se pondrá la textura de las de **hojas oscuras**, mientras que para la **pirámide central**, las **hojas claras**.

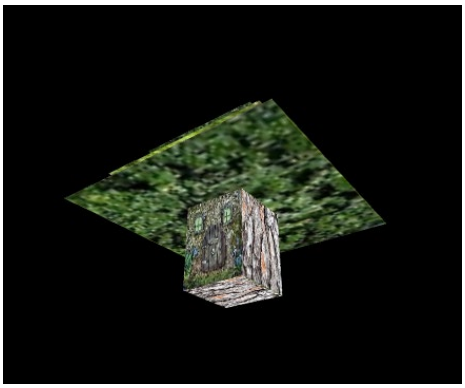
Cara frontal



Caras laterales

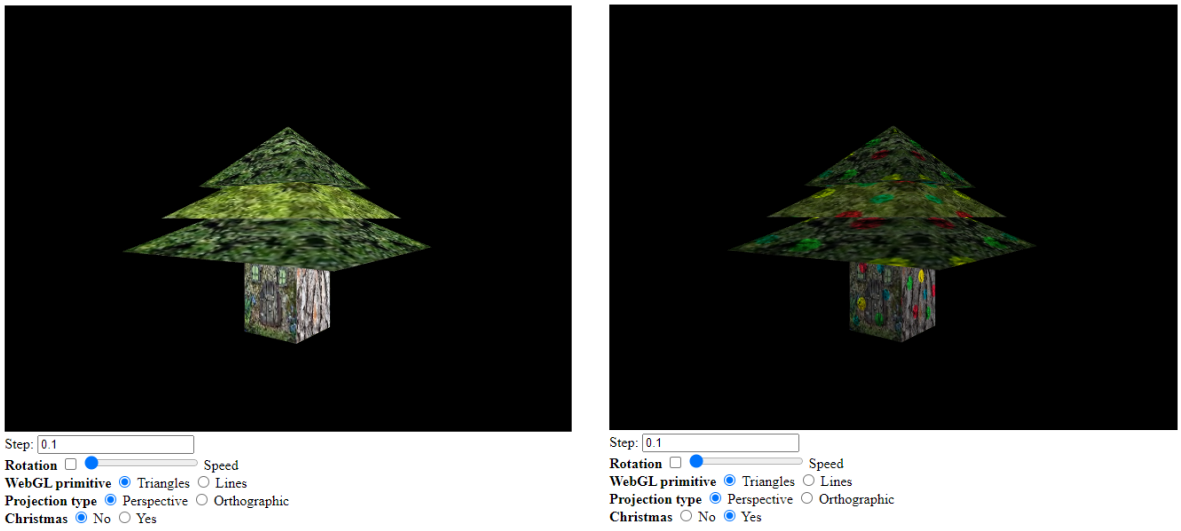


Pirámides



Cara inferior

2. Añade un **radio button** que permita elegir entre dos opciones para **Christmas: No / Yes**, siendo **No** la opción elegida por defecto. Cuando el usuario elija la opción **Yes**, se aplicará la **segunda imagen "light.jpg"** a la textura del árbol con la textura del apartado anterior. Cuando vuelva a elegir **No**, se dejará de aplicar esta segunda imagen y la textura será la original del árbol añadida en el apartado anterior.



3. Se capturan las pulsaciones de teclado en la página web, de modo que se realizarán las siguientes transformaciones sobre la cámara en función de la tecla pulsada y del valor **Step** configurable desde la interfaz de usuario:
    - **Tecla Flecha arriba:** Elevar la cámara mirando al árbol.
    - **Tecla Flecha abajo:** Descender la cámara mirando al árbol.

Ayuda

Los movimientos de la cámara están asociados con la función `glMatrix.mat4.lookAt()`.

Puedes incluir los nuevos controles necesarios en la interfaz de usuario como sigue:

```
<body onload="init()">
    […]
    <b>Christmas</b>
    <input type="radio" name="christmas" value="no" checked> No
    <input type="radio" name="christmas" value="yes"> Yes<br>
</body>
```

Ejercicio 7 – Iluminación con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 7 de la asignatura “Iluminación con WebGL”.

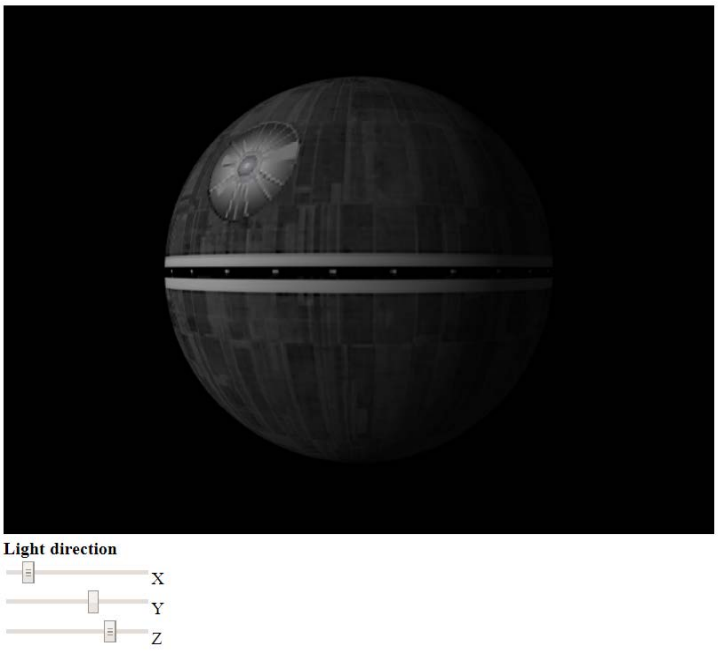
Como resultado de tu práctica deberás generar un **único fichero comprimido .zip** que deberás subir al Aula Virtual, que contendrá **un fichero HTML** y la **imagen** de textura.

Puntos totales posibles del ejercicio: 10

Instrucciones

Partiendo del ejemplo visto en clase “esfera con textura” del tema 6 (texturas), se pide:

1. **Cambiar la textura** por una nueva (”deathstar.png”).
2. Añadir iluminación por **luz direccional** (reflexión difusa) y **ambiente** calculada **por fragmento** (degradado realista). La luz **direccional** tendrá un **color blanco (1.0, 1.0, 1.0)** y la luz **ambiente** también, pero reducida en sus tres componentes **(0.1, 0.1, 0.1)**.
3. Las coordenadas de la dirección de la luz (X, Y, Z) se podrán cambiar mediante botones de rango en la interfaz de usuario de la página web.



Ayuda

```
<body onload="init()">
   <canvas id="myCanvas" width="640" height="480"></canvas><br>
   <b>Light direction</b><br>
   <input type="range" id="x" min="-20" max="20" value="-15" step="1">X<br>
   <input type="range" id="y" min="-20" max="20" value="5" step="1">Y<br>
   <input type="range" id="z" min="-20" max="20" value="10" step="1">Z<br>
</body>
```

Ejercicio 7 – Iluminación con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 7 de la asignatura “Iluminación con WebGL”.

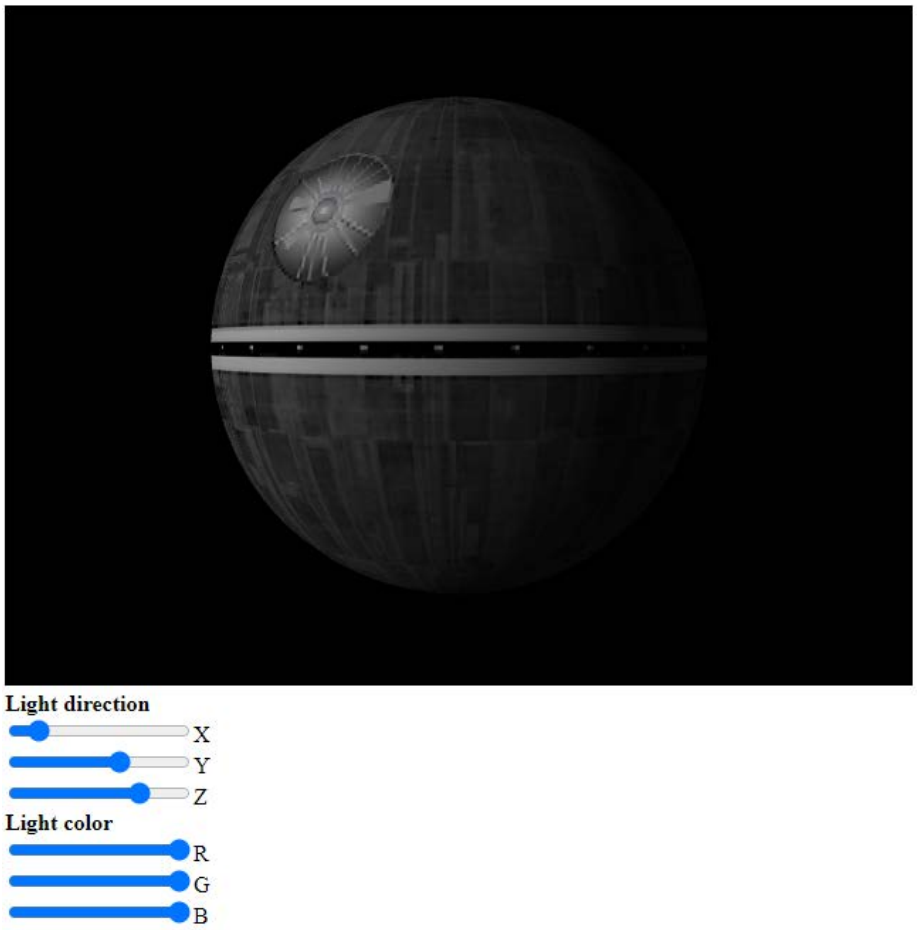
Como resultado de tu práctica deberás generar un **único fichero comprimido .zip** que deberás subir al Aula Virtual, que contendrá **un fichero HTML** y la **imagen** de textura.

Puntos totales posibles del ejercicio: 10

Instrucciones

Partiendo del ejemplo visto en clase “esfera con textura” del tema 6 (texturas), se pide:

1. **Cambiar la textura** por una nueva (”deathstar.png”).
2. Añadir iluminación por **luz direccional** (reflexión difusa) y **ambiente** calculada **por fragmento** (degradado realista). La luz **direccional** tendrá un **color** seleccionable mediante botones de rango **(R, G, B)** y la luz **ambiente** tendrá un **color blanco**, pero reducida en sus tres componentes **(0.1, 0.1, 0.1)**.
3. Las coordenadas de la dirección de la luz (X, Y, Z) se podrán cambiar mediante botones de rango en la interfaz de usuario de la página web.



Ayuda

```
<body onload="init()">
   <canvas id="myCanvas" width="640" height="480"></canvas><br>
   <b>Light direction</b><br>
   <input type="range" id="x" min="-20" max="20" value="-15" step="1">X<br>
   <input type="range" id="y" min="-20" max="20" value="5" step="1">Y<br>
   <input type="range" id="z" min="-20" max="20" value="10" step="1">Z<br>
   <b>Light color</b><br>
   <input type="range" id="r" min="0" max="1" value="1" step="0.1">R<br>
   <input type="range" id="g" min="0" max="1" value="1" step="0.1">G<br>
   <input type="range" id="b" min="0" max="1" value="1" step="0.1">B<br>
</body>
```

Ejercicio 7 – Iluminación con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 7 de la asignatura “Iluminación con WebGL”.

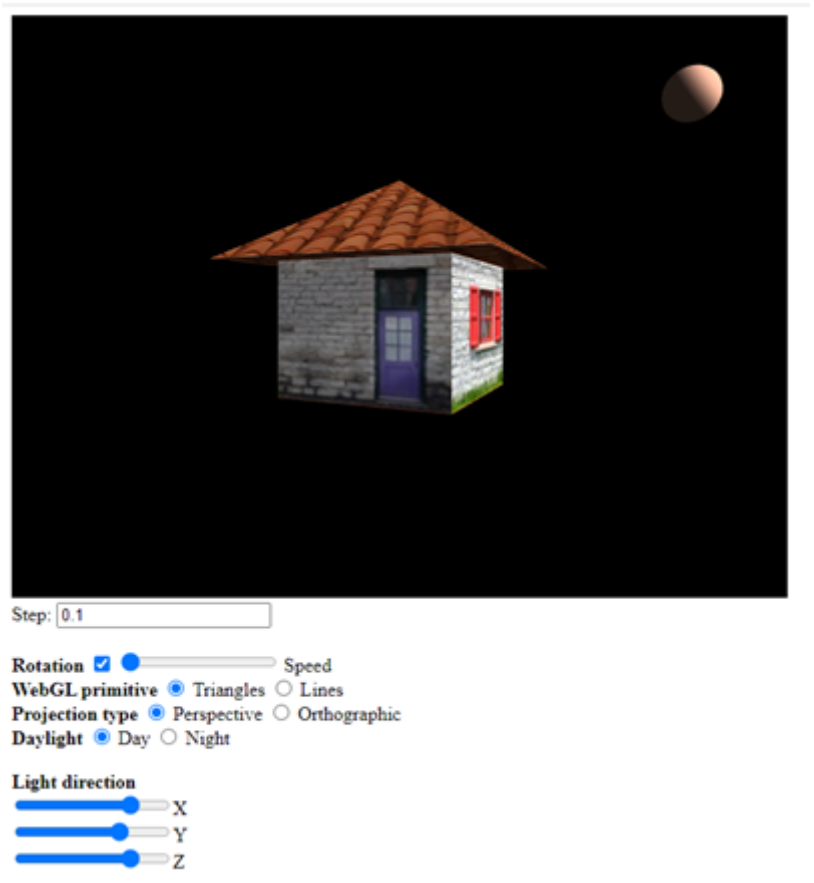
Como resultado de tu práctica deberás generar un **único fichero comprimido .zip** que deberás subir al Aula Virtual, que contendrá **un fichero HTML** y la **imagen** de textura.

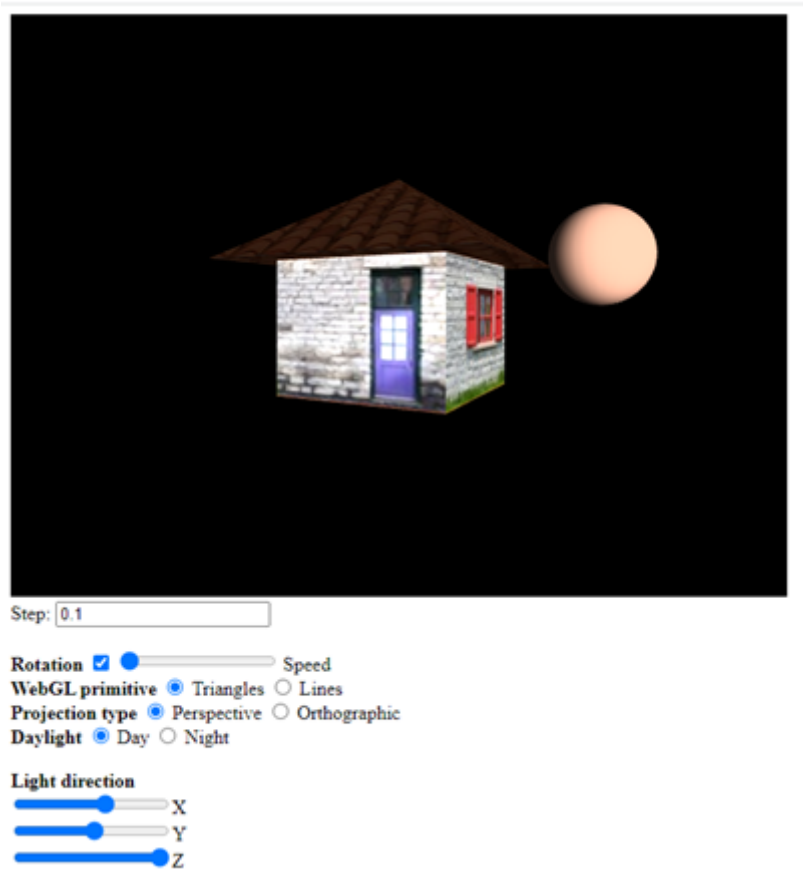
Puntos totales posibles del ejercicio: 10

Instrucciones

Partiendo del ejercicio anterior, se pide hacer las siguientes modificaciones:

1. Añadir iluminación por **luz direccional** (reflexión difusa) y **ambiente**, calculada **por fragmento** (degradado realista). La luz **direccional** tendrá un **color blanco**, y la luz **ambiente** tendrá también un **color blanco**, pero reducida en sus tres componentes **(0.2, 0.2, 0.2)**. Para el tejado, considerar que la orientación de sus vértices es vertical.
2. Las **coordenadas** de la dirección de la luz (X, Y, Z) se podrán cambiar mediante botones de rango en la interfaz de usuario de la página web.
3. Pese a que los rayos en la luz direccional provienen del infinito independientemente de la distancia, **dibujar una esfera** de radio 0.1 situada en la posición (X, Y, Z) elegida por el usuario. Esta esfera deberá verse afectada igualmente por la luz que le incide y su textura es a elección del diseñador, a elegir dentro de la textura facilitada.





Ayuda

```
<body onload="init()">
   <canvas id="myCanvas" width="640" height="480"></canvas><br>
   <b>Light direction</b><br>
   <input type="range" id="x" min="-2" max="2" value="1.1" step="0.1">X<br>
   <input type="range" id="y" min="-2" max="2" value="0.8" step="0.1">Y<br>
   <input type="range" id="z" min="-2" max="2" value="1.1" step="0.1">Z<br>
</body>
```

Ejercicio 7 – Iluminación con WebGL

Este ejercicio tiene como objetivo implementar una aplicación WebGL poniendo en práctica todos los conceptos estudiados en el tema 7 de la asignatura “Iluminación con WebGL”.

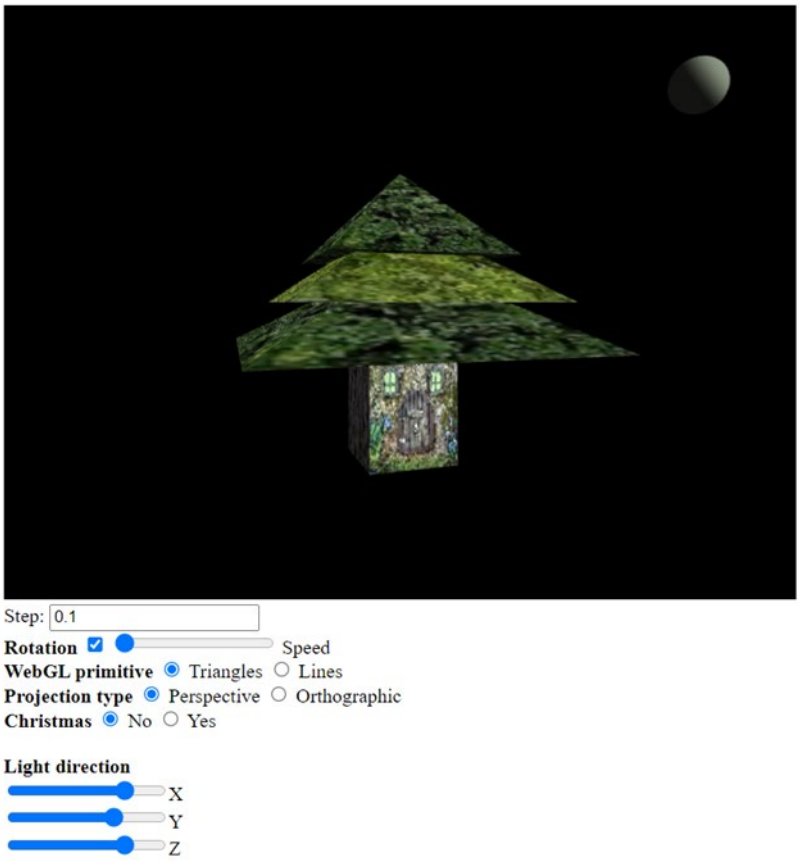
Como resultado de tu práctica deberás generar un **único fichero comprimido .zip** que deberás subir al Aula Virtual, que contendrá **un fichero HTML** y las **imágenes** de textura.

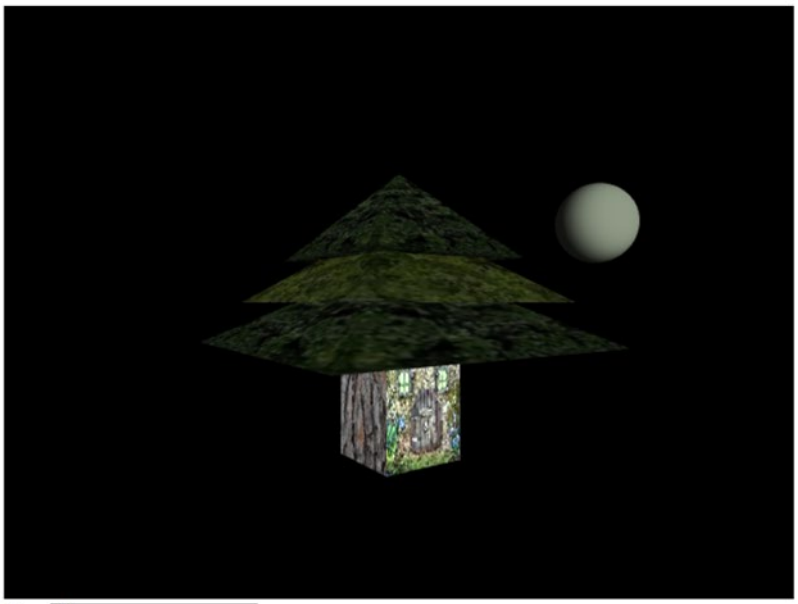
Puntos totales posibles del ejercicio: 10

Instrucciones

Partiendo del ejercicio anterior, se pide hacer las siguientes modificaciones:

1. Añadir iluminación por **luz direccional** (reflexión difusa) y **ambiente**, calculada **por fragmento** (degradado realista). La luz **direccional** tendrá un **color blanco**, y la luz **ambiente** tendrá también un **color blanco**, pero reducida en sus tres componentes **(0.2, 0.2, 0.2)**. Para las hojas, considerar que la orientación de sus vértices es vertical.
2. Las **coordenadas** de la dirección de la luz (X, Y, Z) se podrán cambiar mediante botones de rango en la interfaz de usuario de la página web.
3. Pese a que los rayos en la luz direccional provienen del infinito independientemente de la distancia, **dibujar una esfera** de radio 0.1 situada en la posición (X, Y, Z). Esta esfera deberá verse afectada igualmente por la luz que le incide, y su textura es a elección del diseñador, a elegir dentro de las texturas facilitadas.





Step: 0.1

Rotation Speed

WebGL primitive Triangles Lines

Projection type Perspective Orthographic

Christmas No Yes

Light direction

X

Y

Z

Ayuda

```
<body onload="init()">
   <canvas id="myCanvas" width="640" height="480"></canvas><br>
   <b>Light direction</b><br>
   <input type="range" id="x" min="-2" max="2" value="1.1" step="0.1">X<br>
   <input type="range" id="y" min="-2" max="2" value="0.8" step="0.1">Y<br>
   <input type="range" id="z" min="-2" max="2" value="1.1" step="0.1">Z<br>
</body>
```

Ejercicio 8 – Conceptos básicos en Three.js

El objetivo de este ejercicio es empezar a familiarizarse con la librería de gráficos 3D Three.js.

No hay que subir ninguno de los ficheros realizados en cada parte al Aula Virtual.

Puntos totales posibles del ejercicio: 0

Parte 1 – Geometrías

1. Abre la documentación oficial de Three.js, concretamente la parte de geometrías

https://threejs.org/docs/index.html#api/en/geometries/BoxBufferGeometry

2. Examina los diferentes tipos de geometrías. Usa los controles de los ejemplos para cambiar los parámetros.
3. Incorpora diferentes tipos de geometrías (por ejemplo, un cubo o una esfera) en una página web local que use Three.js.

Ejercicio 9 – Geometrías y texturas en Three.js

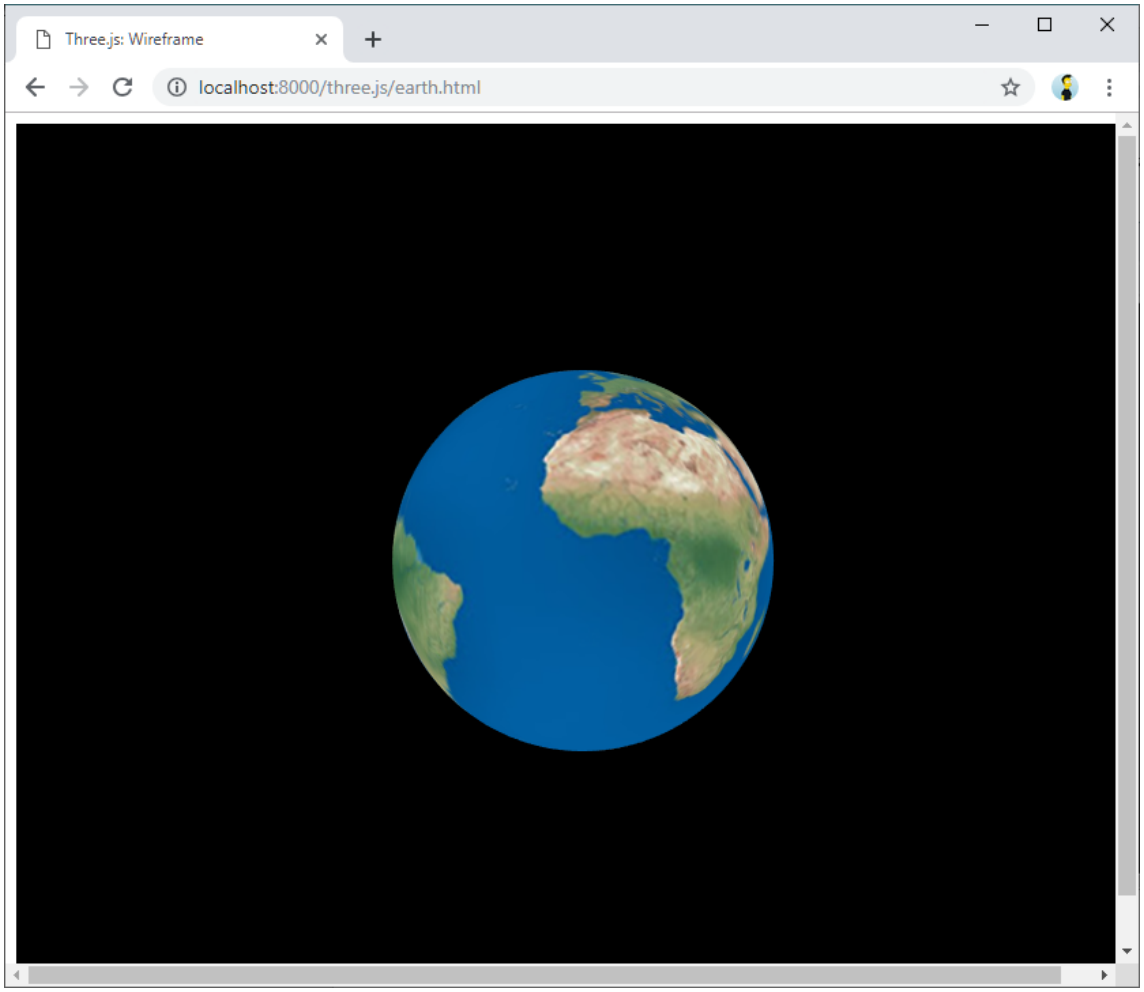
El objetivo de este ejercicio es empezar a familiarizarse con la librería de gráficos 3D Three.js.

No hay que subir ninguno de los ficheros realizados en cada parte al Aula Virtual.

Puntos totales posibles del ejercicio: 0

Parte 1 – Geometrías y texturas

1. Genera el gráfico de la esfera con textura que vimos en WebGL, pero esta vez usando Three.js.



Práctica Final – Pong en Three.js

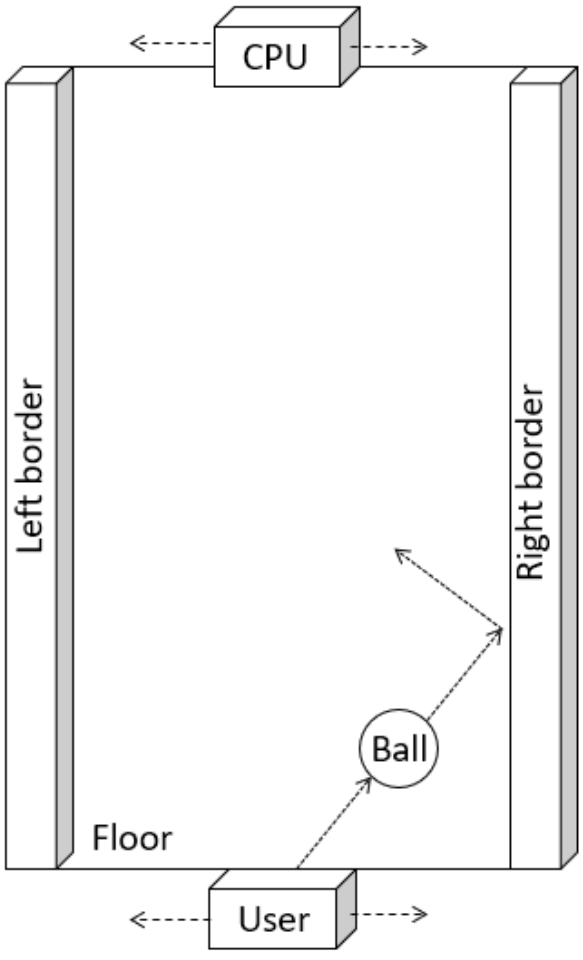
El objetivo de esta práctica es desarrollar una aplicación web que contenga un gráfico 3D interactivo creado con Three.js. Este gráfico implementará el juego clásico: **Pong**.

Puntos totales posibles del ejercicio: 10

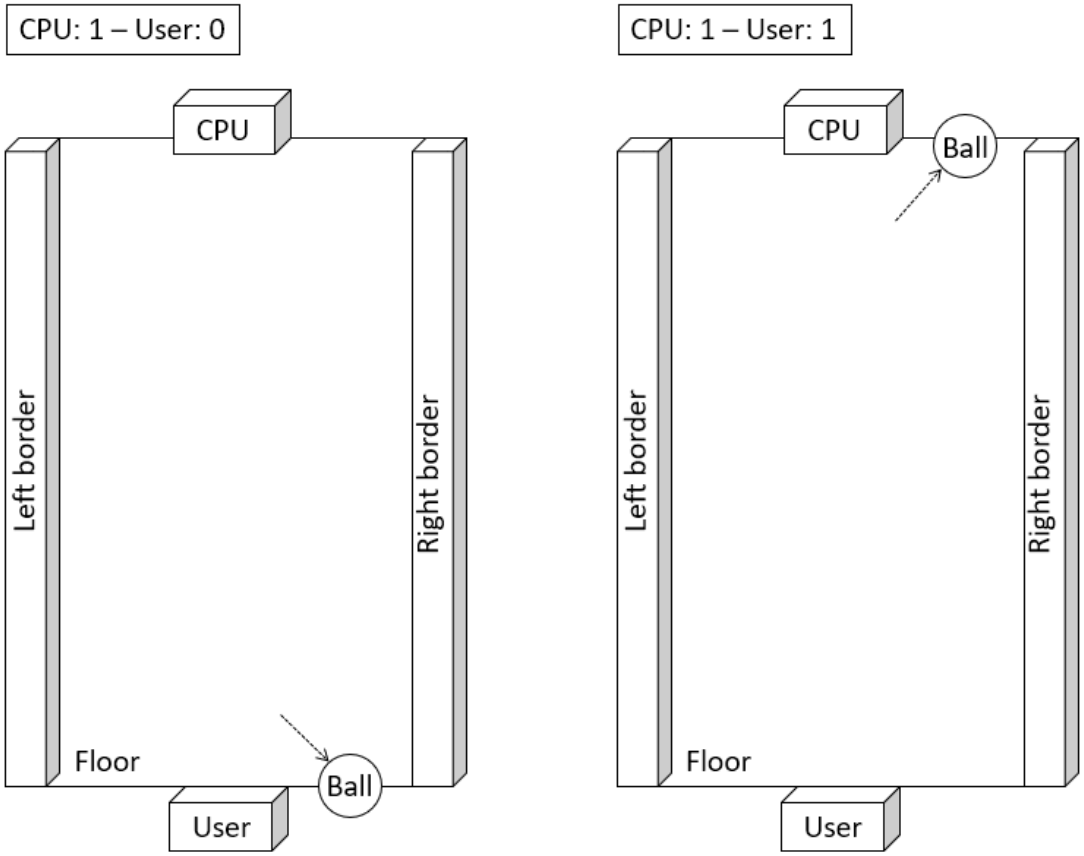
Funcionamiento básico

El gráfico 3D que implemente el juego estará formado por los siguientes elementos:

1. Suelo. Superficie plana en 2D por la que se desplazará la bola.
2. Bola. Esfera en 3D que se utilizará en el desarrollo del juego.
3. Bordes izquierdo y derecho. Hexaedros en los laterales del plano de juego (suelo).
4. CPU. Hexaedro situado en la parte superior del plano. Su movimiento en el eje X será automático.
5. Usuario. Hexaedro situado en la parte inferior del plano. Su movimiento en el eje X será realizado por el jugador, capturando las pulsaciones de teclado de las flechas izquierda y derecha.



La bola se desplazará por encima del suelo variando la posición de sus coordenadas X e Y. Existirá detección de colisión de la bola con respecto a los bordes derecho e izquierdo, así como con respecto a los hexaedros de CPU y usuario. El juego consistirá en devolver la bola por parte de la CPU y el usuario, desplazando los respectivos hexaedros hasta bloquear el paso de la bola en el momento en el está en la parte superior e inferior del plano respectivamente. Si el hexaedro de CPU/usuario bloquean esta posición, la bola rebotará en la dirección contraria. En caso contrario, la bola saldrá del plano y se añadirá un punto al jugador (CPU o usuario) que haya conseguido hacer caer la bola por el lado contrario. Por ejemplo:



En algún lugar de la página web deberá existir un marcador que muestre la puntuación. En el funcionamiento básico, no es necesario que sea dentro del gráfico Three.js (se puede hacer con elementos HTML de tipo texto). Se empezará con 0-0 y el primer jugador en llegar a 5 puntos será el vencedor.

Al cargar la página el juego no empezará hasta que el usuario pulse una tecla (por ejemplo, la barra espaciadora). En ese momento la bola empezará a moverse en una de las 2 direcciones del plano (hacia CPU o usuario). El hexaedro de la CPU se moverá automáticamente siguiente la trayectoria de la coordenada X de la esfera, mientras que el hexaedro del usuario se moverá la izquierda o derecha en función de la pulsación de las teclas ← y → del teclado. Cuando CPU o usuario marquen un punto el juego se parará y se actualizará el marcador. Al llegar a 5 se declarará ganador a uno de los jugadores y se ofrecerá la opción de volver a jugar.

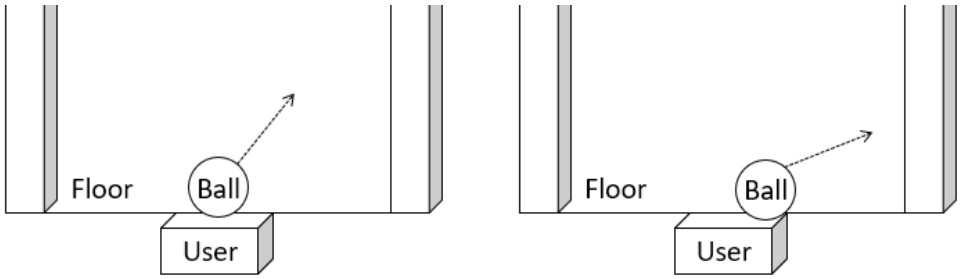
El modo de funcionamiento básico supone que se usarán los siguientes elementos:

- Texturas para todos los objectos (suelo, bordes, CPU, usuario, y bola).
- Fuente de luz direccional o puntual. Además, La bola deberá producir efectos de sombra, que será reflejada tanto en el suelo, como en los bordes y hexaedros (CPU/usuario).

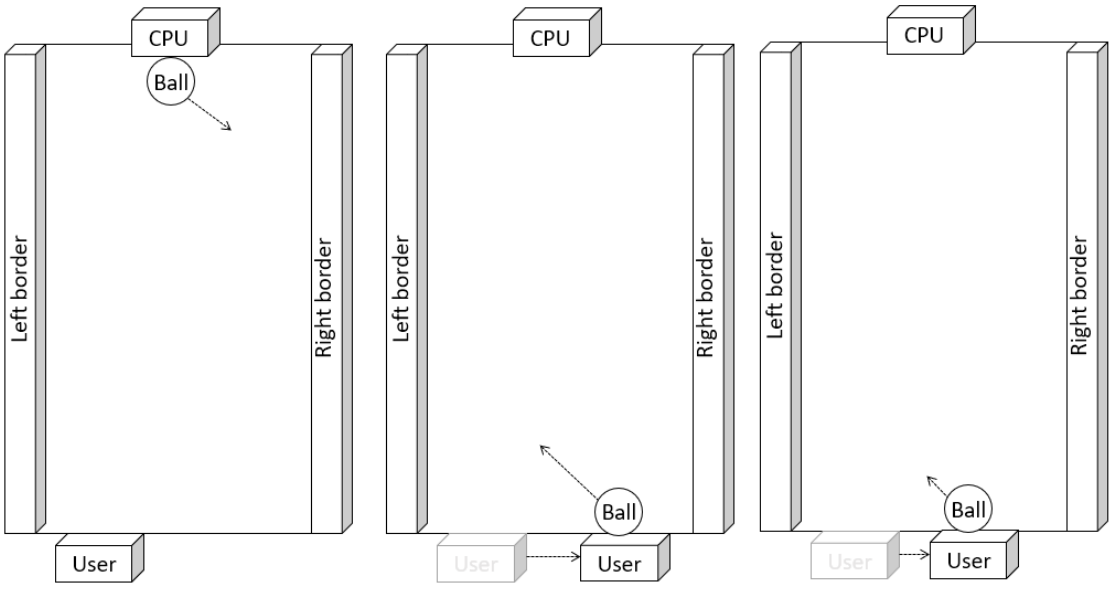
Funcionamiento avanzado

En el funcionamiento básico se supone que la velocidad con la que la bola se desplaza verticalmente (esto es, como se mueve a lo largo del eje Y) y el ángulo con el que rebota en los bordes y jugadores (esto es, como se mueve a lo largo eje X) es constante. En el funcionamiento avanzado estos 2 parámetros serán variables.

Para variar el **ángulo de rebote**, se tendrá en cuenta la posición del hexaedro con respecto a la bola a la hora de bloquearla. El ángulo será mayor si se bloquea con el extremo del hexaedro, y menor si se bloquea de forma centrada.



Para la **velocidad de desplazamiento vertical** se tendrá en cuenta el desplazamiento del hexaedro de su posición inicial (cuando el jugador contrario rebota la pelota) respecto al punto donde se bloquea. A mayor desplazamiento horizontal, mayor velocidad de desplazamiento vertical.



Tanto en el modo básico como en el modo avanzado se da por supuesto que la CPU va a poder bloquear siempre la bola. Para que sea un juego más realista, debería existir la **posibilidad de poder ganar a la CPU**. Para ello, habría que configurar adecuadamente la velocidad con la que la CPU se desplaza horizontalmente, de modo que existan situaciones en las que no le dé tiempo a llegar a bloquear la bola. Se puede incluso introducir una componente de aleatoriedad en el desplazamiento del hexaedro de la CPU para conseguir este efecto.

Mejoras

Se deja abierto una parte de calificación (ver sección evaluación) para introducir mejoras extras en la práctica. Las posibilidades de mejora pueden ser variadas. Se calificarán las mejoras que mejoren la experiencia de uso de la aplicación. Algunas ideas son:

- Introducir niveles de dificultad. Pueden ser configurable al iniciar el juego o ir aumentando según se juego una y otra vez.
- Marcador con gráfico 3D en lugar de como texto HTML.
- Posibilidad de elegir diferentes texturas para los elementos de juego (“skin”).
- Animaciones adicionales (movimiento de la cámara o punto de luz en función del juego).
- …

Para optar a la nota máxima (10 puntos) habrá que implementar al menos dos mejoras extras.

Evaluación

Esta práctica se evaluará con un máximo de 10 puntos. La distribución de estos puntos será de la siguiente manera:

- Funcionamiento básico: 5 puntos.
- Funcionamiento avanzado: 3 puntos.
- Mejoras: 2 puntos.

Además, se tendrá en cuenta la calidad de código. Se supone que el código deberá cumplir los siguientes principios:

1. Se deben evitar los fragmentos de código duplicados (principio DRY, Don’t Repeat Yourself)
2. El código debe ser legible. Esto se traduce en:
    a. Usar nombres de variables, funciones, etc. significativos (preferiblemente en inglés).
    b. El código debe estar correctamente indentado (usando tabs o espacios).
    c. No se utilizará código innecesario (por ejemplo, funciones que no se usan, condiciones que no se cumplen, etc).

Si se detectan problemas de calidad de código en base a estos dos principios, se restará hasta 1 punto de la nota obtenida al sumar las componentes anteriores (funcionamiento básico, avanzado, y mejoras).

Consejos

Es aconsejable seguir las siguientes recomendaciones:

- Comenzar implementando la funcionalidad básica. Una vez realizada, si se aspirar a sacar más de 5 puntos, se puede implementar la funcionalidad avanzada y por último las mejoras extras.
- Ir paso a paso. La mejor estrategia sería Implementar pequeños cambios comprobando verificando en cada momento que todo funciona como se espera.
- Usar la consola de depuración del navegador (Chrome, Firefox) para visualizar las trazas en la consola JavaScript (console.log) así como los posibles errores según vamos desarrollando.

Entrega

La entrega se realizará de forma **individual**. Hay que subir un archivo comprimido .zip en el aula virtual con todos los ficheros de la práctica, esto es:

- Página web (fuente HTML).
- Imagen(es) de textura.
- Librerías JavaScript. Si se usa código JavaScript fuera de la página HTML (aunque no es necesario que el código JavaScript vaya en un fichero aparte, puede ir embebido dentro de la página web).

La práctica será evaluada en primer lugar ejecutada desde un navegador Chrome y servida con un servidor web local.

En la cabecera de la página web deberá aparecer un comentario (esto es, un párrafo entre los símbolos `<!--` y `-->`) informando de las partes que has implementado. Por ejemplo:

```
<!--

Partes implementadas:

- Funcionalidad básica
- Funcionalidad avanzada (ángulo de rebote variable, velocidad de
desplazamiento vertical variable, posibilidad de ganar a la CPU)

 -->
<!DOCTYPE html>
<html>

<head>
<title>Pong</title>
```

El ejemplo anterior aspiraría a sacar una máxima de 8 puntos (5 de la funcionalidad básica y 3 de la funcionalidad avanzada).

Práctica Final – Breakout en Three.js

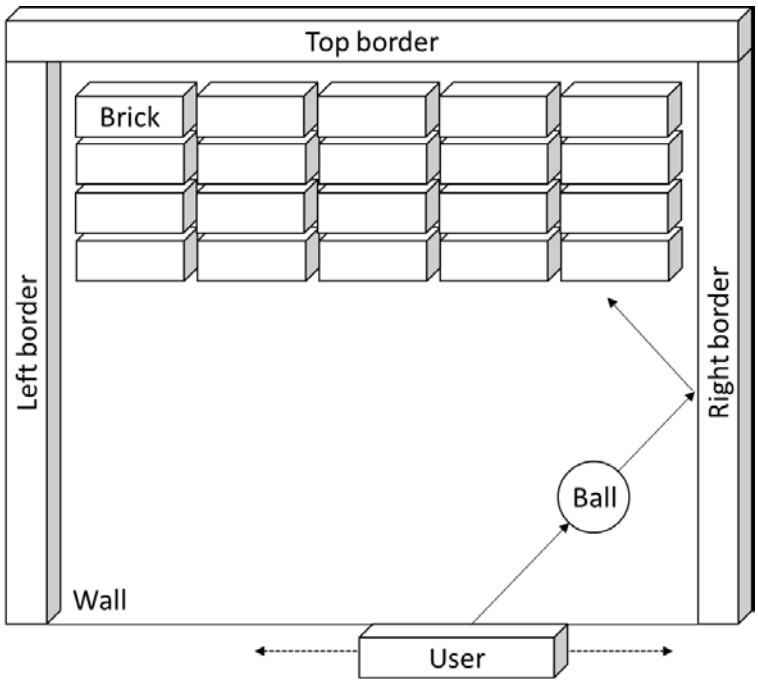
El objetivo de esta práctica es desarrollar una aplicación web que contenga un gráfico 3D interactivo creado con Three.js. Este gráfico implementará el juego clásico: **Breakout**.

Puntos totales posibles del ejercicio: 10

Funcionamiento básico

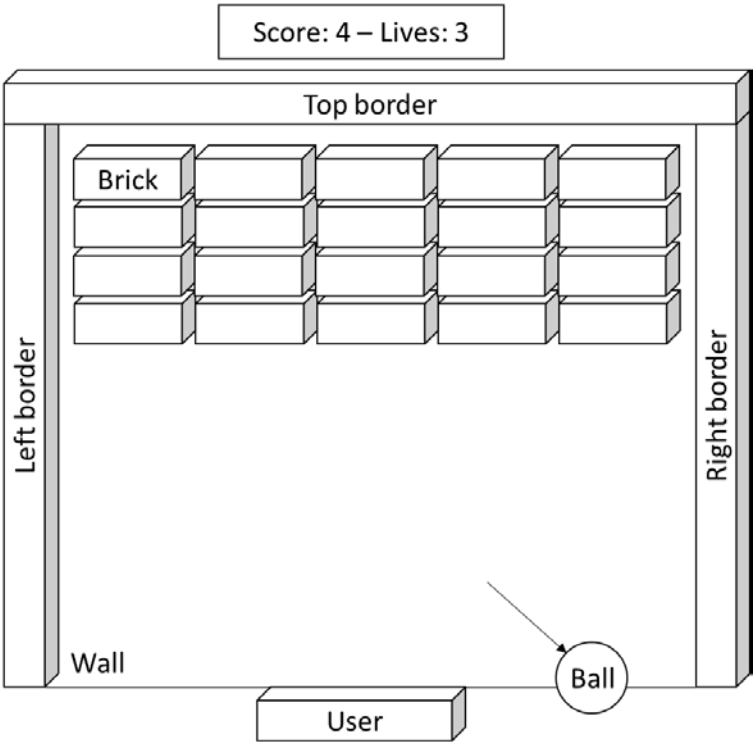
El gráfico 3D que implemente el juego estará formado por los siguientes elementos:

1. Pared. Superficie plana en 2D por la que se desplazará la bola.
2. Bola. Esfera en 3D que se utilizará en el desarrollo del juego.
3. Borde izquierdo, derecho y superior. Hexaedros en los laterales del plano de juego (pared).
4. Ladrillos: Hexaedros situados en la zona de juego (5-7 columnas y 4-5 filas).
5. Usuario. Hexaedro situado en la parte inferior del plano. Su movimiento en el eje X será realizado por el jugador, capturando las pulsaciones de teclado de las flechas izquierda y derecha.



La bola se desplazará por la pared variando la posición de sus coordenadas X e Y. Existirá detección de colisión de la bola con respecto a los bordes derecho, izquierdo y superior, así como con respecto a los hexaedros de usuario y los ladrillos. El juego consistirá en devolver la bola por parte del usuario, desplazando el respectivo hexaedro, hasta que todos los ladrillos hayan sido eliminados. Si el hexaedro de usuario bloquea esta posición, la bola rebotará en la dirección contraria. En caso contrario, la bola saldrá del plano y se restará una vida al jugador.

Por ejemplo:



En algún lugar de la página web deberá existir un marcador que muestre la puntuación y el número de vidas. En el funcionamiento básico, no es necesario que sea dentro del gráfico Three.js (se puede hacer con elementos HTML de tipo texto). Se empezará con 0 puntos y 3 vidas, y se sumará un punto por cada bloque eliminado y se restará una vida por cada bola perdida. Si el jugador se queda sin vidas o elimina todos los bloques, el juego finalizará.

Al cargar la página el juego no empezará hasta que el usuario pulse una tecla (por ejemplo, la barra espaciadora). En ese momento la bola empezará a moverse hacia arriba y en una de las 2 direcciones del plano (izquierda o derecha). El hexaedro del usuario se moverá a la izquierda o derecha en función de la pulsación de las teclas ← y → del teclado. Siempre que el usuario elimine un bloque o pierda una vida se actualizará el marcador, y en caso de perder una vida, además el juego se parará hasta que el usuario pulse de nuevo la tecla de inicio. Al perder las 3 vidas o al eliminar todos los bloques, se declarará ganador (si ha eliminado todos los ladrillos) o perdedor (si pierde todas sus vidas) y se ofrecerá la opción de volver a jugar.

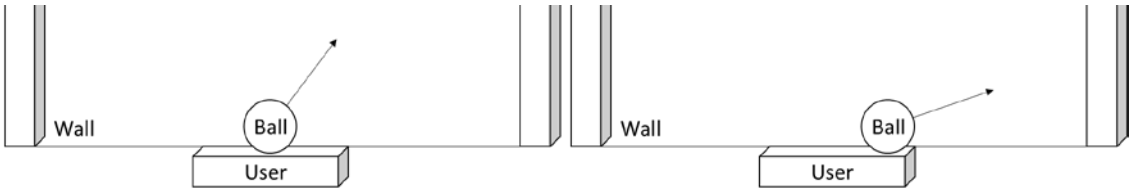
El modo de funcionamiento básico supone que se usarán los siguientes elementos:

- Texturas para todos los objectos (pared, bordes, ladrillos, usuario y bola).
- Fuente de luz direccional o puntual. Además, La bola deberá producir efectos de sombra, que será reflejada tanto en la pared, como en los bordes y hexaedros.

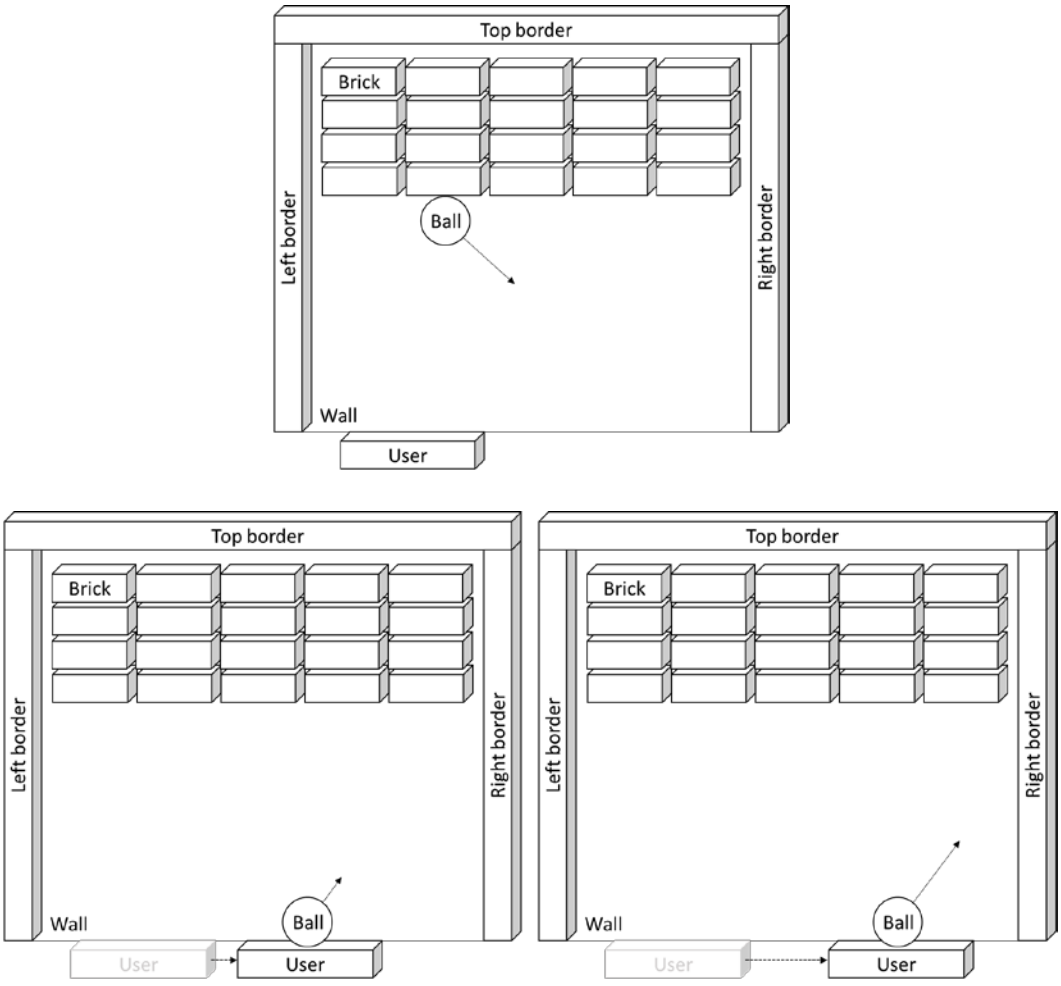
Funcionamiento avanzado

En el funcionamiento básico se supone que la velocidad con la que la bola se desplaza verticalmente (esto es, como se mueve a lo largo del eje Y) y el ángulo con el que rebota en los bordes, ladrillos y jugadores (esto es, como se mueve a lo largo eje X) es constante. En el funcionamiento avanzado estos 2 parámetros serán variables.

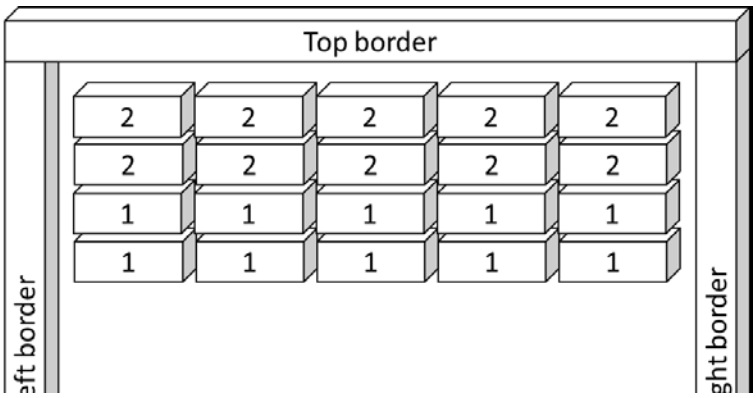
Para variar el **ángulo de rebote**, se tendrá en cuenta la posición del hexaedro con respecto a la bola a la hora de bloquearla. El ángulo será mayor si se bloquea con el extremo del hexaedro, y menor si se bloquea de forma centrada.



Para la **velocidad de desplazamiento vertical** se tendrá en cuenta el desplazamiento del hexaedro de su posición inicial (cuando rebota la bola sobre un ladrillo) respecto al punto donde se bloquea por el usuario. A mayor desplazamiento horizontal, mayor velocidad de desplazamiento vertical.

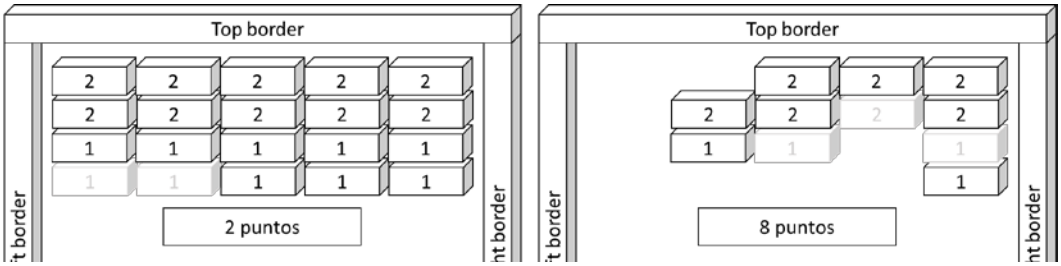


Además, para que sea un juego más interesante, se tendrá en cuenta el número de bloques eliminado en cada jugada y su posición. De este modo, cada bloque que inicialmente tiene un valor de puntuación de 1, tendrá un valor distinto dependiendo de su posición vertical. La fila más cercana al usuario, tendrá un valor de 1, el cuál se verá incrementado en 1 unidad por cada dos filas (a más filas, más puntuación).



Además, si el número de bloques eliminados entre un rebote de la bola con el usuario y el siguiente es mayor que 3, la puntuación obtenida en esa jugada valdrá el doble.

Por ejemplo:



Mejoras

Se deja abierto una parte de calificación (ver sección evaluación) para introducir mejoras extras en la práctica. Las posibilidades de mejora pueden ser variadas. Se calificarán las mejoras que mejoren la experiencia de uso de la aplicación. Algunas ideas son:

- Introducir niveles de dificultad. Pueden ser configurables al iniciar el juego o ir aumentando según se juega una y otra vez.
- Marcador con gráfico 3D en lugar de como texto HTML.
- Posibilidad de elegir diferentes texturas para los elementos de juego (“skin”).
- Animaciones adicionales (movimiento de la cámara o punto de luz en función del juego).
- …

Para optar a la nota máxima (10 puntos) habrá que implementar al menos dos mejoras extras.

En caso de duda sobre las mejoras que serán aceptadas, antes de implementar nada, siempre consultar con el profesor para tener el visto bueno.

Evaluación

Esta práctica se evaluará con un máximo de 10 puntos. La distribución de estos puntos será de la siguiente manera:

- Funcionamiento básico: 5 puntos.
- Funcionamiento avanzado: 3 puntos.
- Mejoras: 2 puntos.

Además, se tendrá en cuenta la calidad de código. Se supone que el código deberá cumplir los siguientes principios:

1. Se deben evitar los fragmentos de código duplicados (principio DRY, Don't Repeat Yourself)
2. El código debe ser legible. Esto se traduce en:
    a. Usar nombres de variables, funciones, etc. significativos (preferiblemente en inglés).
    b. El código debe estar correctamente indentado (usando tabs o espacios).
    c. No se utilizará código innecesario (por ejemplo, funciones que no se usan, condiciones que no se cumplen, etc).

Si se detectan problemas de calidad de código en base a estos dos principios, se restará hasta 1 punto de la nota obtenida al sumar las componentes anteriores (funcionamiento básico, avanzado, y mejoras).

Consejos

Es aconsejable seguir las siguientes recomendaciones:

- Comenzar implementando la funcionalidad básica. Una vez realizada, si se aspira a sacar más de 5 puntos, se puede implementar la funcionalidad avanzada, y por último las mejoras extras.
- Ir paso a paso. La mejor estrategia sería implementar pequeños cambios comprobando y verificando en cada momento que todo funciona como se espera.
- Usar la consola de depuración del navegador (Chrome, Firefox) para visualizar las trazas en la consola JavaScript (console.log) así como los posibles errores según vamos desarrollando.

Entrega

La entrega se realizará de forma **individual**. Hay que subir un archivo comprimido .zip en el aula virtual con todos los ficheros de la práctica, esto es:

- Página web (fuente HTML y JavaScript).
- Imagen(es) de textura.
- Fuentes (estilo CSS, texto JSON, sonido MP3, etc.)

La práctica será evaluada en primer lugar ejecutada desde un navegador Chrome y servida con un servidor web local.

En la cabecera de la página web deberá aparecer un comentario (esto es, un párrafo entre los símbolos <!-- y -->) informando de las partes que has implementado y tu nombre. Por ejemplo:

```
<!--
Autor: José Miguel

Partes implementadas:

- Funcionalidad básica
- Funcionalidad avanzada:
      - Ángulo de rebote variable
      - Velocidad de desplazamiento vertical variable
- Mejoras:
      - Niveles de dificultad

 -->
<!DOCTYPE html>
<html>

<head>
<title>Breakout</title>
```

El ejemplo anterior aspiraría a sacar una máxima de 8 puntos (5 de la funcionalidad básica, 2 de la funcionalidad avanzada y 1 de las mejoras).

Universidad Rey Juan Carlos

Escuela de Ingeniería de Fuenlabrada