Universidad Rey Juan Carlos

Gráficas y Visualización 3D

Práctica 6: *Matrix*

Katia Leal Algara

Agustín Santos Méndez

1. **Uso de matrices**: “Example 4-1.html”, este ejercicio es similar a los ejercicios anteriores, pero en lugar de mover el objeto utilizando un escalar (un float) vamos a utilizar una matriz. El uso de vectores y matrices en WebGL es fundamental ya que nos permite mover, escalar o rotar los objetos en el espacio.

   En este primer ejemplo, empezamos introduciendo una matriz que nos permitirá mover el triángulo de una forma un poco más interesante en el futuro. Por ahora lo moveremos igual que antes, lateralmente. Comparad esto con los ejercicios anteriores.

   Lo primero que tenemos que observar es que hemos incluido una librería de javascript llamada “**gl-matrix-min.js**”. Nos servirá para crear y modificar matrices.

   El siguiente punto más relevante es la creación de una variable llamada “uMVMatrix”. Esta es de tipo *uniform* pero esta vez en lugar de ser un float, se trata de una matriz. Eso es lo que indica la palabra “**mat4**”. Ahora usamos las propiedades de las matrices para cambiar la posición del triangulo en el eje “X”.

   Finalmente, el cambio que nos permite utilizar las matrices aparece cuando tratamos con las variables uniform desde Javascript. Con “mat4.identity” reconvertimos la matriz a un estado inicial correspondiente a la matriz identidad. Y con “mat4.translate” hacemos que la matriz contenga la información necesaria para hacer una translación.

Ejercicios:

a) Sin cambiar el vertex shader modifica la construcción de la matriz para que el cambio afecte al eje “Y”. Es decir que en lugar de moverse de derecha a izquierda se mueva de arriba a abajo.
b) Sin cambiar el vertex shader modifica la construcción de la matriz para que se mueva a la vez de derecha a izquierda y de arriba a abajo.
c) Sin cambiar el vertex shader modifica la construcción de la matriz para que el triángulo se mueva haciendo círculos.

2. **Escalado**: “Example 4-2.html”, Este ejercicio es muy similar al anterior pero esta vez usamos la matriz para escalar el triángulo. Observa que no modificamos los shaders. Solo la forma de construir la matrix. De hecho, solo cambiamos una sola línea. Esa es la potencia de usar matrices.

   Ejercicios:

   a) Sin cambiar el vertex shader modifica la construcción de la matriz para que el escalado afecte solo al eje “Y”.
   b) Dibuja dos triángulos, uno que se escale en el eje “X” y el otro que se escale en el eje “Y”.
   c) Combina el ejercicio anterior para que además se muevan en círculos. Es decir combina un escalado con una traslación. Ten mucho cuidado con el orden de cálculo. No es lo mismo “trasladar y escalar” que “escalar y trasladar”.

3. **Rotación**: “Example 4-3.html”, Este ejercicio es muy similar al anterior pero esta vez usamos la matriz para rotar el triángulo. Al igual que nos sucedía antes, observa que no modificamos los shaders. Solo la forma de construir la matrix. De nuevo vemos en juego la potencia de usar matrices.

   Ejercicios:

   a) Sin cambiar el vertex shader modifica la construcción de la matriz para que la rotación afecte solo al eje “X”.
   b) Sin cambiar el vertex shader modifica la construcción de la matriz para que la rotación afecte tanto al eje “X” como al eje “Y”
   c) Dibuja dos triángulos, uno que rote en sentido horario y el otro en el sentido contrario.

4. **Combinando**: “Example 4-4.html”, en este ejemplo jugamos con varios triángulos que cambian de forma y posición independientemente. Pero a pesar de que el resultado parece complejo realmente estamos usando el mismo modelo y los mismos shaders para el dibujo. La única diferencia es que cada triangulo construye la matriz de modelo de una forma diferente.

   La principal diferencia con los ejemplos anteriores es que ahora antes de dibujar el triángulo se prepara una matriz diferente.

   Ejercicios:

   a) Dibuja cuadrados pequeños que se muevan por pantalla con movimientos independientes.
   b) Dibuja un cuadrado pequeño en el centro y haz que dos o tres cuadrados se muevan a su alrededor. Cada cuadrado debe tener un tamaño diferente (pero utiliza el mismo buffer para todos). La idea es simular un sistema planetario de tu invención.
   c) (Avanzado). Las técnicas de “flocking” inventadas por Craig Reynolds en 1986, son muy interesantes e instructivas. Hay multitud de implementaciones realizadas en Java o Javascript. Y el material explicando el algoritmo es inmenso (https://en.wikipedia.org/wiki/Boids). Intenta hacer algo parecido usando los ejemplos que hemos realizado hasta la fecha con WebGL. Pero hazlo sencillo. Los “boids” serán triángulos.

      Boid: A computer simulation of an animal that flies in *flocks* or *swarms*.

      Un ejemplo de lo que podría lo que buscamos lo tienes en http://www.paulboxley.com/blog/2012/05/boids. Pero tu implementación debe usar únicamente lo que hemos visto hasta la fecha.