Mapeo de textura

Joseba Makazaga

UPV/EHU

Se pretende implementar una aplicación que mapee una textura sobre un triángulo. El triangulo se define mediante 3 puntos tridimensionales (x, y, z), donde $x, y \in (0, 500)$ y $z \in (-250, 250)$. A cada punto hay que asignarle las coordenadas de textura que vienen dadas mediante los valores (u, v) donde $u, v \in (0, 1)$.

La textura a utilizar es una foto en formato **ppm**, que se cargará en la aplicación desde el fichero **foto.ppm**.

La aplicación debe leer desde el fichero **triangles.txt** la definición de los triangulos que se definan en el fichero: cada línea del fichero que comience con el caracter **t** define un triangulo mediante 15 números, cinco para cada vértice del triángulo; los tres primeros representan las coordenadas (x, y, z), coordenadas espaciales del triángulo, y los otros dos representan las coordenadas de textura (u, v) que definen la parte de la foto que hay que mapear sobre el triángulo.

Además de leer la información de los triángulos la aplicación debe dibujar en cada momento un triángulo con la textura mapeada. El triángulo a dibujar se puede cambiar mediante la tecla <**ENTER**>.

Al principio se debe dibujar el priemer triángulo y cada vez que se pulse la tecla hay que pasar al siguiente triángulo. Si estamos dibujando el último, el siguiente será el primero. La aplicación terminará cuando se pulse <**ESC**>.

Por ejemplo, dada la foto de la Figura 1, si tenemos un triangulo en

Figure 1: Foto utilizada para mapear la textura

Figure 2: Mapeo de la parte inferior izquierda sobre la parte inferior izquierda del lienzo

la parte inferior izquierda de la ventana de dibujo, y mapeamos sobre ese triangulo la parte inferior izquierda de la foto, es decir, dados los puntos con las coordenadas indicadas en la siguiente tabla:

	Coordenadas	textura
Punto 1	$(0, 0, 0)$	$(0, 0)$
Punto 2	$(500, 0, 0)$	$(1, 0)$
Punto 3	$(0, 500, 0)$	$(0, 1)$

obtendremos la imagen mostrada en la Figura 2:

Pero podemos mapear la misma parte de la foto sobre un triángulo rotado y escalado ligeramente, es decir, mantenemos las coordenadas de textura pero hacemos más pequeño y rotamos el triángulo donde mapeamos la foto. Si las coordenadas de los tres puntos del triángulo son: punto 1 en $(20, 20, 0)$, punto 2 en $(480, 0, 0)$ y punto 3 en $(40, 480, 0)$, obtendremos la imagen mostrada en la izquierda de la Figura 3. En la imagen de la derecha de la misma figura se puede ver la misma parte de textura mapeada sobre un triángulo aún más pequeño y mas rotado.

Las tres imágenes de las Figuras 2 y 3 mantienen las proporciones de la foto y el triángulo del lienzo (ambos son triángulos isosceles y rectángulos), pero podemos hacer que la forma del triangulo dibujado y la parte de la textura que mapeamos sobre el triángulo tengan diferentes formas, por ejemplo si utilizamos las mismas coordenadas de textura que en los anteriores dibujos (parte inferior izquierda de la foto) pero lo mapeamos sobre tiángulos que no son isosceles, como los mostrados en la Figura 4, se puede ver cómo se deforma el dibujo mapeado.

En realidad podemos sobreponer cualquier triángulo de la foto sobre cualquier triángulo del lienzo, como se puede obesrvar en la Figura 5.

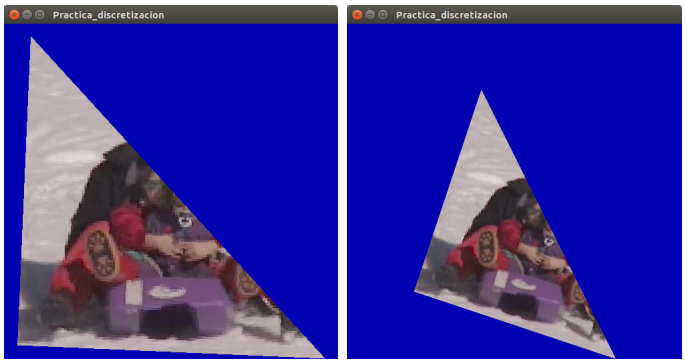

Figure 3: Mapeo de la parte inferior izquierda de la foto sobre dos triangulos, los dos rotados (el de la derecha con una mayor rotación) y escalados (derecha menor tamaño) en el lienzo

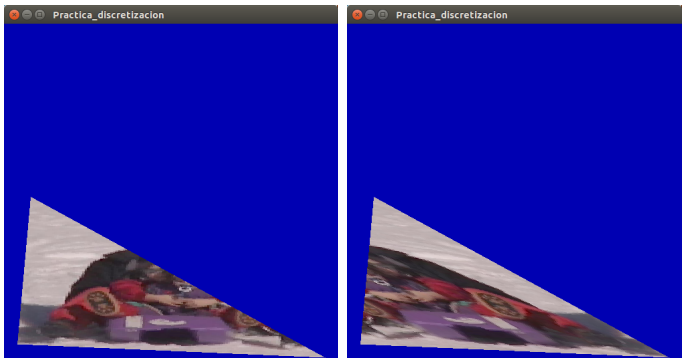

Figure 4: Mapeo de la parte inferior izquierda del tablero sobre dos triángulos con distintas formas en el lienzo, izquierda aplastado derecha invertido en coordenada u

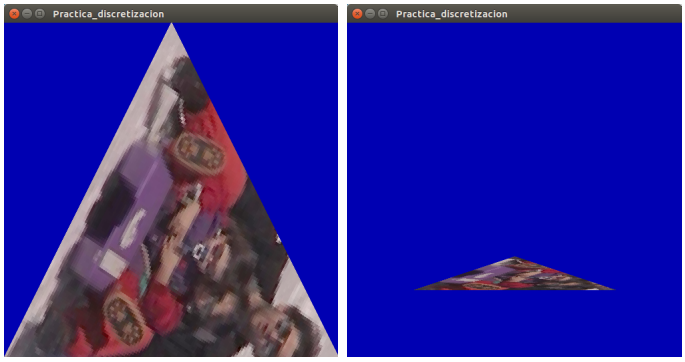

Figure 5: Mapeo de diferentes triángulos de la textura sobre sobre cualquier triángulo en el lienzo

Para facilitar la implementación de la aplicación se dispone del código **cargar-triangulos.c** donde se define la funciçon que lee desde el fichero **triangles.txt** las coordenadas de los tres vértices de cada triángulo así como las coordenadas de textura u,v asociadas a cada vértice,

Para cargar la foto (debe ser una foto en formato ppm) disponemos del fichero cargar-ppm.c donde se define la función que carga en un buffer y en las variables dimx y dimy la información de la foto (el fichero que contiene la foto ha de llamarse **foto,ppm**).

Además, disponemos de la aplicación inicial **dibujar-puntos.c** que muestra la forma de dibujar en un lienzo los puntos que queramos. En ese código el color que se le asigna a cada pixel es el color corresponde al primer pixel de la foto.

Para poder implementar la aplicación conviene tener una función que dadas las coordenadas (u, v) de la textura calcule el color que le corresponde:

unsigned char *color_textura(u,v)

Esta función devuelve el puntero a tres caracteres que definen el color (r, g, b) del pixel más cercano a las coordenadas (u, v) en la foto.

Además de la función que calcula el color de textura que corresponde a las coordenadas de textura, hay que implementar la función que partiendo de la información del triángulo (15 números) calcule todos los puntos internos al triángulo:

dibujar_triangulo(...)

Dicha función debe recibir la información de un triángulo que se habrá leido del fichero y que tiene la información de los tres puntos (cuyas coordenadas

son $(x1, y1, z1)$ para el primer punto, $(x2, y2, z2)$ para el segundo... Además al punto 1 le corresponde el punto $(u1, v1)$ de la foto, al punto 2 le corresponde el punto $(u2, v2)$ de la foto...). Para cada punto interno del triángulo hay que calcular sus coordenadas (x, y, z) y las coordenadas (u, v). Con las coordenadas (x, y, z) sabemos donde debemos dibujar el pixel, y con las coordenadas (u, v) podremos asignar el color al pixel.