Programación de Videojuegos

Universidad Autónoma de San Luis Potosí
Facultad de Ingeniería
Parcial 1
Angel de Jesús Maldonado Juárez
Actividad 1
17 de febrero del 2023

Generación de texturas

La generación de texturas en la programación gráfica conlleva establecer las coordenadas en las cuales se agregarán los gráficos o información de una imagen, en esta actividad se le agrega al `VBO` (Vertex Buffer Object), pares de coordenadas a cada punto, las cuales representan las posiciones a partir de las cuales se van a mostrar las texturas:

```
float vertices[] = {                                                    T1                                  T2
    0.5f, 0.5f, 0.0f,   1.0f, 0.0f, 0.0f,   1.0f,1.0f,   3.0f, 3.0f,
    0.5f, -0.5f, 0.0f,  0.0f, 1.0f, 0.0f,   1.0f,0.0f,   3.0f, 0.0f,
    -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f,   0.0f,0.0f,   0.0f, 0.0f,
    -0.5f, 0.5f, 0.0f,  1.0f, 1.0f, 0.0f,   0.0f,1.0f,   0.0f, 3.0f
};
```

Posteriormente, para almacenar la información de las texturas, como el id de la textura, ruta del archivo de imagen, y la cantidad de texturas se crearon las siguientes variables:

```
const int nTextures = 2;
unsigned int textures[nTextures];

string texturesFiles[] = {
    "Imagenes/caja.jpg",
    "Imagenes/fragil.png"
};
```

Para generalizar la generación de los buffers de las texturas se agregaron los siguientes parámetros a la función `GeneraTextura()`:

```
void GeneraTextura(int tipo, unsigned int* textura, const char* archivo)
```

- `tipo`: indica si el archivo de imagen es *.png* o *.jpg* (0 o 1 respectivamente).
- `textura`: es la variable que guarda el id de la textura.
- `archivo`: es la ruta del archivo de imagen de la textura.

Una vez que el archivo de imagen fue cargado correctamente con la función `stbi_load()`, se definen distintos parámetros para las texturas con base en el tipo de imagen que se cargó (0 para png y 1 para jpg).

```
switch (tipo) {
    // PNG
    case 0:
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
        glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);

        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, data);
        break;
    // JPG
    case 1:
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
        break;
}

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glGenerateMipmap(GL_TEXTURE_2D);
stbi_image_free(data);
```

Posteriormente, una vez que hayan sido cargadas las texturas al buffer, en la función `updateWindow()` se actualizan las variables uniformes `texture0` y `texture1` para que el *fragmentShader* pueda distinguir ambas texturas (con las locaciones 0 y 1 en el buffer), después se activan y se enlazan ambas texturas con `active()` y `bind()`:

```
glUniform1i(glGetUniformLocation(ourShader.ID, "texture0"), 0);
glUniform1i(glGetUniformLocation(ourShader.ID, "texture1"), 1);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, textures[0]);

glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, textures[1]);
```

Finalmente, el *fragmentShader* utiliza la función `mix()` para combinar ambas texturas y lograr que la transparencia del archivo png pueda permitir que la imagen de abajo se logre distinguir:

```
void main()
{
    //FragColor = texture(texture0, ourTexture); //* vec4(ourColor, 1.0f);
    FragColor = mix(texture(texture0, ourTexture), texture(texture1, ourTexture), 0.3f);
}
```

Al ejecutar el código se obtiene el siguiente resultado: