Universidad Nacional
Autónoma de México

Facultad de Ingeniería

Ingeniería en Computación

Práctica 4
Terrenos y múltiples texturas

Alumno(a)
Isabel Gómez Yareli Elizabeth
Ortiz Figueroa María Fernanda
Roldan Rivera Luis Ricardo

Asignatura
Computación Gráfica Avanzada

Grupo
1

Profesor
M.C. Reynaldo Martell Ávila

Fecha de entrega
5 de marzo de 2020

2020 - 2

Objetivo

Diseñar e implementar una imagen en escala de grises y una imagen en escala de negro - RGB para crear un terreno con diferentes alturas y textura a través de un BlendMapColor , en un proyecto de Visual Studio con OpenGL que cuente con modelos 3D animados.

Desarrollo

Para realizar esta práctica se hizo uso de diversas texturas con el objetivo de dar un enfoque más realista a escenario de nuestro proyecto al emplear un blend, el cual nos permitirá generar una mezcla entre diferentes texturas para poder observar un degradado por ejemplo entre un camino de tierra y el pasto en las orillas de este.

Ahora bien, la manera en la que es posible implementar diversas texturas, es al tener como referencia un mapa con 4 colores (negro, rojo, verde y azul), en donde a cada uno le corresponderá una textura diferente, sin embargo, no solo basta con lo anterior, en el fragment shader del terreno es necesario indicar la relación entre las coordenadas de la textura y el mapa de texturas de colores.
Para poder agregar las diferentes texturas es necesario realizar la asignación y suma de cada uno de los colores, así como la definición de cada uno de ellos.

```
.fs*  main.cpp
63  uniform sampler2D blendMapTexture;
64
65  vec3 calculateDirectionalLight(Light light, vec3 direction){
66      vec2 tiledCoords = our_uv;
67      if(tiledCoords.x != 0 && tiledCoords.y != 0)
68          tiledCoords = scaleUV * tiledCoords;
69
70      // Referencia con respecto al mapa de multiples texturas y la coordenadas UV
71      vec4 blendMapColor = texture(blendMapTexture, our_uv);
72      // Obtención del color respecto al mapa en escala de negro - RGB
73      float backTextureAmount = 1 - (blendMapColor.r + blendMapColor.g + blendMapColor.b);
74      // Textura color negro -> arena
75      vec4 backgroundTextureColor = texture(backgroundTexture, tiledCoords) * backTextureAmount;
76      // Textura color rojo -> piedra
77      vec4 rTextureColor = texture(rTexture, tiledCoords) * blendMapColor.r;
78      // Textura color verde -> musgo
79      vec4 gTextureColor = texture(gTexture, tiledCoords) * blendMapColor.g;
80      // Textura color azul -> agua
81      vec4 bTextureColor = texture(bTexture, tiledCoords) * blendMapColor.b;
82      vec4 totalColor = backgroundTextureColor + rTextureColor + gTextureColor + bTextureColor;
```

Posteriormente, en el main del proyecto es necesario realizar diversas configuraciones:

- Variables GLuint correspondiente a cada una de las texturas y el mapa a emplear, así como la variable Terrain para indicar el mapa en escala de grises correspondiente a las diversas alturas que podremos observar en la escena de nuestro proyecto.

```
89  // Terrain model instance
90  Terrain terrain(-1, -1, 200, 8, "../Textures/heightmap.png");
91
92  GLuint textureCespedID, textureWallID, textureWindowID, textureHighwayID, textureLandingPadID;
93
94  GLuint textureTerrainBackgroundID, textureTerrainRID, textureTerrainGID, textureTerrainBID, textureTerrainBlendMapID;
```

- Se inicializa el *shader*

```
// Inicialización de los shaders
shader.initialize("../Shaders/colorShader.vs", "../Shaders/colorShader.fs");
shaderSkybox.initialize("../Shaders/skyBox.vs", "../Shaders/skyBox.fs");
shaderMulLighting.initialize("../Shaders/iluminacion_textura_animation.vs", "../Shaders/multipleLights.fs");
shaderTerrain.initialize("../Shaders/terrain.vs", "../Shaders/terrain.fs");
```

- Se inicializa el objeto correspondiente al terreno.

```
242
243     terrain.init();
244     terrain.setShader(&shaderTerrain);
245     terrain.setPosition(glm::vec3(100, 0, 100));
```

- Se definen cada una de las texturas a emplear, las cuales corresponden a imágenes en formato PNG, de tal forma que se requiere especificar a cada una de ellas la ubicación, indice, filtros, el formato de la imágen y los datos mismos de la textura.

```
489     // Definiendo la textura a utilizar
490     // Texture textureTerrainBackground("../Textures/grassy2.png");
491     // Color NEGRO en el mapa de texturas
492     Texture textureTerrainBackground("../Textures/arena.png");
493     // Carga el mapa de bits (FIBITMAP es el tipo de dato de la libreria)
494     bitmap = textureTerrainBackground.loadImage();
495     // Convertimos el mapa de bits en un arreglo unidimensional de tipo unsigned char
496     data = textureTerrainBackground.convertToData(bitmap, imageWidth, imageHeight);
497     // Creando la textura con id 1
498     glGenTextures(1, &textureTerrainBackgroundID);
499     // Enlazar esa textura a una tipo de textura de 2D.
500     glBindTexture(GL_TEXTURE_2D, textureTerrainBackgroundID);
501     // set the texture wrapping parameters
502     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
503     // set texture wrapping to GL_REPEAT (default wrapping method)
504     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
505     // set texture filtering parameters
506     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
507     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
508     // Verifica si se pudo abrir la textura
509     if (data) {
510         // Transferir los datos de la imagen a memoria
511         // Tipo de textura, mipmaps, formato interno de openGL, ancho, alto, mipmaps, formato
512         // interno de la libreria de la imagen, el tipo de dato y al apuntador a los datos
513         glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, imageWidth, imageHeight, 0,
514         GL_BGRA, GL_UNSIGNED_BYTE, data);
515         // Generan los niveles del mipmap (OpenGL es el ecargado de realizarlos)
516         glGenerateMipmap(GL_TEXTURE_2D);
517     } else
518         std::cout << "Failed to load texture" << std::endl;
519     // Libera la memoria de la textura
520     textureTerrainBackground.freeImage(bitmap);
```

- Después, se indica la destrucción y borrado de los objetos empleados para el terreno, así como las texturas mismas.

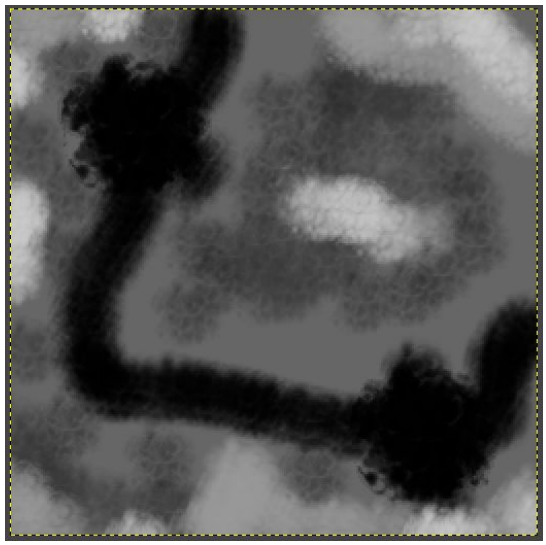
```
701     // Textures Delete
702     glBindTexture(GL_TEXTURE_2D, 0);
703     glDeleteTextures(1, &textureCespedID);
704     glDeleteTextures(1, &textureWallID);
705     glDeleteTextures(1, &textureWindowID);
706     glDeleteTextures(1, &textureHighwayID);
707     glDeleteTextures(1, &textureLandingPadID);
708     glDeleteTextures(1, &textureTerrainBackgroundID);
709     glDeleteTextures(1, &textureTerrainRID);
710     glDeleteTextures(1, &textureTerrainGID);
711     glDeleteTextures(1, &textureTerrainBID);
712     glDeleteTextures(1, &textureTerrainBlendMapID);
```

- Se indican las respectivas configuraciones del terreno con respecto al terreno y se activan cada una de las texturas, en donde es necesario especificar el número de shader, la unidad de textura y el objeto textura, ya que hay una correlación directa para poder visualizar cada una de ellas de forma idónea en el escenario.

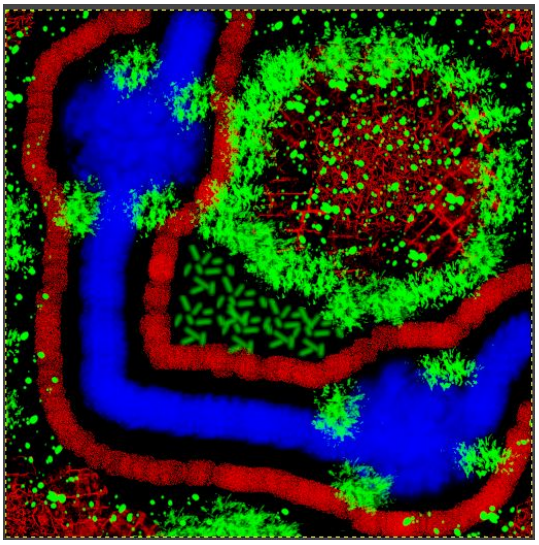
```
964     /*******************************************
965      * Propiedades SpotLights
966      *******************************************/
967     shaderMulLighting.setInt("spotLightCount", 0);
968     shaderTerrain.setInt("spotLightCount", 0);
969
970     /*******************************************
971      * Propiedades PointLights
972      *******************************************/
973     shaderMulLighting.setInt("pointLightCount", 0);
974     shaderTerrain.setInt("pointLightCount", 0);
975
976     /*******************************************
977      * Terrain Cesped
978      *******************************************/
979     glm::mat4 modelCesped = glm::mat4(1.0);
980     modelCesped = glm::translate(modelCesped, glm::vec3(0.0, 0.0, 0.0));
981     modelCesped = glm::scale(modelCesped, glm::vec3(200.0, 0.001, 200.0));
982     // Se activa la textura del background/arena
983     glActiveTexture(GL_TEXTURE0);
984     glBindTexture(GL_TEXTURE_2D, textureTerrainBackgroundID);
985     shaderTerrain.setInt("backgroundTexture", 0);
986     // Se activa la textura de tierra/piedras
987     glActiveTexture(GL_TEXTURE1);
988     glBindTexture(GL_TEXTURE_2D, textureTerrainRID);
989     shaderTerrain.setInt("rTexture", 1);
```

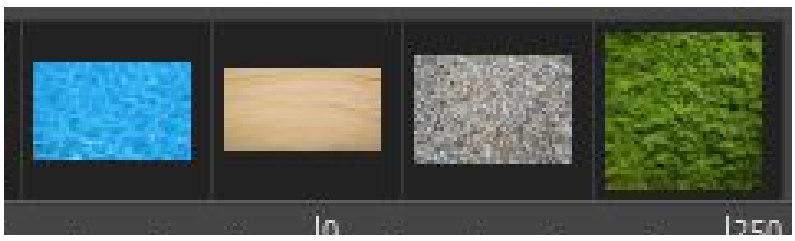
En general, lo anterior son los aspectos a considerar para poder implementar un terreno con múltiples texturas, de tal manera que a continuación las diferentes actividades que se realizaron con respecto a la presente práctica:

- Se editó en GIMP el mapa en escala de grises para crear una concordancia con respecto a mapa de texturas.

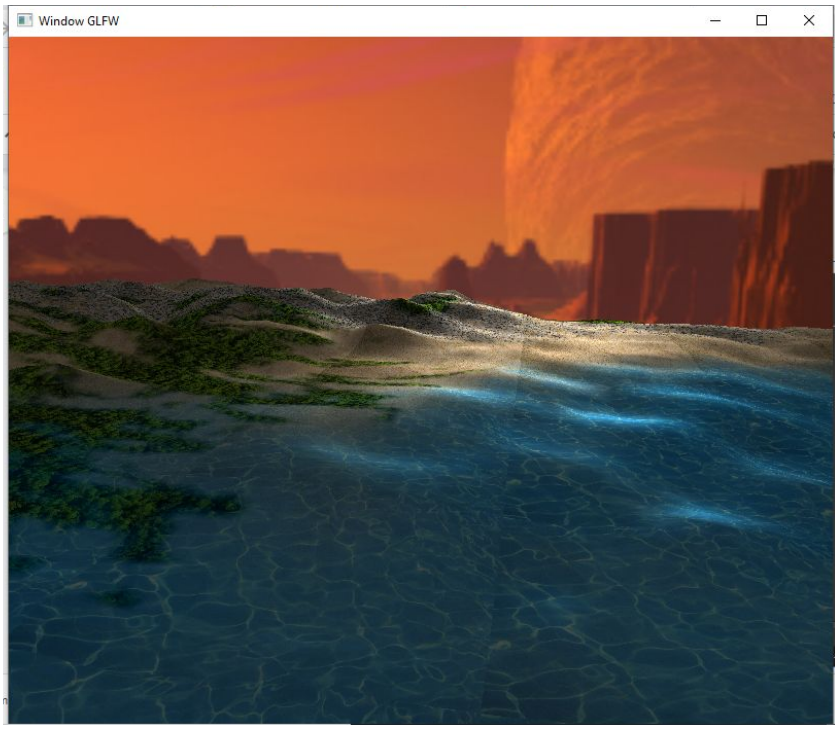


- Se diseñó y creó en GIMP un mapa de texturas dimensiones cuadradas (número potencia de 2), con cada uno de los colores correspondientes a cada textura, es importante señalar que las nuevas texturas a emplear fueron la de arena (negro), piedras (rojo), musgo (verde) y agua (azul).





De tal forma que el resultado de lo anterior fue



- Agregar los modelos 3D animados con los que se había estado trabajando en prácticas pasadas.

Conclusiones

En esta práctica aprendimos a como crear un terreno con relieve (alturas) y múltiples texturas, personalizado con las herramientas de GIMP, en el cual a partir de un fondo negro se dibujan figuras o líneas con en diferentes escalas, según el mapa de referencia generar.

Ahora bien, para poder implementar el mapa en el fragment shader se indicaron los BlendMapColor correspondientes al RGB, a los cuales se les aplicaron operaciones para poderlos ver unificados, lo que nos permitió darnos cuenta de que se debe tener cuidado en esto, ya que podría generarse una resta cuyo valor sea cero, lo que nos representaría un mapa absolutamente negro.

Como en prácticas anteriores lo que usemos se debe inicializar y destruir lo cual se hizo en la variable GlUint correspondiente al blend, además para poder usar las diferentes texturas estas se abrieron, se especificó un índice, formato y filtros aplicados para cada una de ellas, de tal forma que todo esto se pudo integrar con los modelos usados en las prácticas anteriores.

Cabe mencionar, que el procesamiento del escenario ha aumentado, lo que nos permitió darnos cuenta que hay que optimizar los modelos empleados para así disminuir la carga de trabajo.

Repositorio

- https://github.com/Sasfer/CGA2020-2