Triangulo de Sierpinski

Israel Santiago Lopez Cruz
Departamento de Ciencia de la Computación
Universidad Católica San Pablo
Email: israel.lopez@ucsp.edu.pe

April 2020

1. Estructuras de datos utilizadas

Las estructuras usadas fue la clase "Sierpinski"para guardar los triangulos en un vector de triangulos en de tipo "Triangle", la segunda estructura guarda 3 vertices de tipo "Vertex", el cual tiene las coordenadas x, y y z, ademas de los colores RGB, todo esto para mantener un orden.

2. Generacion de subtriangulos

Para generar un nuevo nivel de triangulos tengo en cuenta hasta que triangulo debo renderizar en un determinado nivel (para esto tengo un array de niveles: "levels"); al hacerlo con threads para mayor velocidad debo dividir el proceso entre el numero de nucleos, en createTriangle, hallo los puntos medios de un triangulo y creo los nuevos 3 triangulos. al finalizar este paso debo actualizar el indice donde comienzan los triangulos recien creados, este indice servira para el siguiente nivel porque necesito solo el anterior nivel para el nuevo nivel y no todos los triangulos dentro de mi vector de triangulos, tambien para todo un nivel le asigno un color generado de forma aleatoria, finalmente la funcion reDraw renderiza actualizando el glBufferData con el nivel correspondiente.

```
void newLevel () {
        int newIndex = triangles.size ();
        levels[maxLevel++] = newIndex;
        float red = rand () % 2;
        float green = rand () % 2;
        float blue = rand () % 2;
```

```
        vector<Triangle *> newTriangles = triangles;
        int cores = thread::hardware_concurrency ();
        if ( newIndex - index > cores ) {
                thread *threads = new thread[cores];
                int start = index, end;
                int jump = (newIndex - index) / cores;
                for ( int i = 0; i < cores; ++i ) {
                        end = (i + 1 != cores) ? (start + jump)
                            : newIndex;
                        threads[i] = thread (createTriangle,
                            start, end, red, green, blue,
                            triangles, ref (newTriangles));
                        start = end;
                }
                for ( int i = 0; i < cores; ++i )
                        threads[i].join ();
                delete[] threads;
        } else {
                createTriangle (index, newIndex, red, green,
                    blue, triangles, newTriangles);
        }
        index = newIndex;
        triangles.clear ();
        triangles = newTriangles;
        reDraw ();
        cout << "End doing level, number of triangles: " <<
            triangles.size () << endl;
}
```

3. Coloracion de los triangulos

Para la coloracion se creo 3 shaders, "shader1", utiliza la informacion de cada vertice para su coloracion, dado que cada vertice tiene dentro su color rojo, verde y azul. "shader2", utiliza la informacion de la posicion del vertice para su coloreo, es decir su rojo, verde y azul es su coordenada x, y y z. "shader3", utiliza un uniform para cambiar su color cada segundo, en el loop de renderizacion se halla un rojo, verde y azul segun el tiempo y se le pasa al shader fragment.
Tambien cabe mencionar como funciona cada uno de los shaders utilizados:

- El vertex shader"shader1.vs"manda un vec3 "vertexColor"de los colores del vertice al fragment shader "shader1.fs."el cual asigna el "vertexColor."al "FragColor".

- El vertex shader"shader2.vs"manda un vec3 "vertexColor"de las coordenadas del vertice al fragment shader "shader1.fs.[el] cual asigna el "vertexColor.[a]l "FragColor".

- El fragment shader "shader2.fs"tiene un uniform que se le asigna al "FragColor".

estos archivos deben estar dentro de la carpeta "src", de otro modo debe cambiarse el path.

```
shader1 = new Shader ("../src/shader1.vs", "../src/
    shader1.fs"); /// color esta en los valores del
    vertice como rgb
shader2 = new Shader ("../src/shader2.vs", "../src/
    shader1.fs"); /// los colores son segun la posicion
    de la interfaz
shader3 = new Shader ("../src/shader1.vs", "../src/
    shader2.fs"); /// los colores cambian segun el tiempo
shaderActual = shader1;
```

4. Comandos de teclado o mouse

- Comandos de teclado

  1. tecla .[es]c", termina el programa.
  2. tecla "p", imprime todos los triangulos del nivel renderizado.
  3. tecla "↑", renderiza un nivel mayor al actual.
  4. tecla "↓", renderiza un nivel menor al actual.
  5. tecla "1", activa el shader "shader1".
  6. tecla "2", activa el shader "shader2".
  7. tecla "3", activa el shader "shader3".

- Comandos del mouse

  1. La rueda del mouse hacia arriba/adelante, renderiza un nivel mayor al actual.
  2. La rueda del mouse hacia abajo/atras, renderiza un nivel menor al actual.