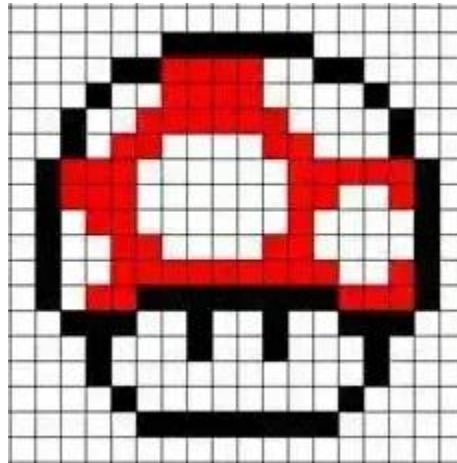
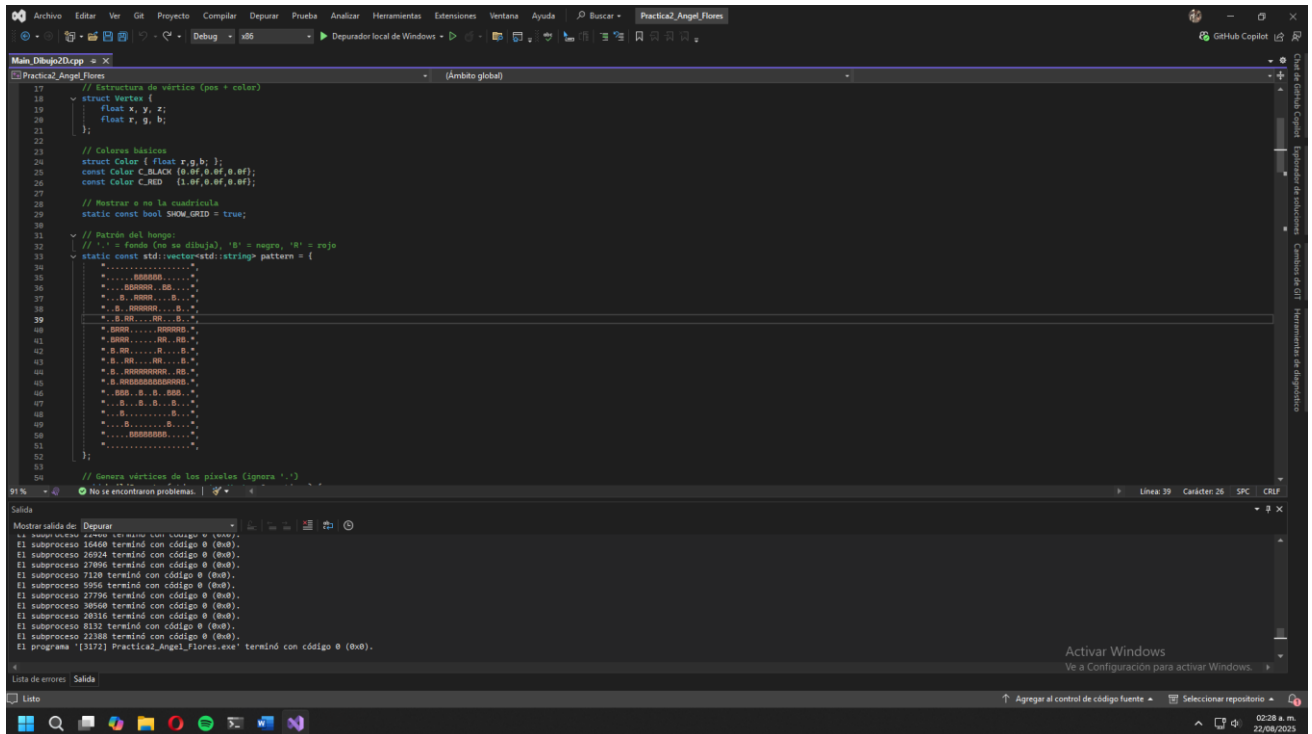


Imagen original de referencia:



Empecé por definir la estructura de mis vértices mediante la posición y el color, y posterior a ello definí dos variables que contuvieran el color negro (C_BLACK) y rojo (C_RED), al principio también tenía definido el color blanco como variables, pero después de seguir avanzando pensé que era más optimo simplemente poner el fondo sin dibujar y utilizarlo para dibujar el hongo, así me ahorra la declaración de otro color, lo único que hice fue asociar un punto (.) al fondo para que cuando lo encontrara no se dibujara.

Una vez definidos los colores y la estructura del vértice declaré una matriz que fui llenando manualmente, en ella añadí la estructura de como se iría dibujando la imagen mediante el uso de los colores asociados. Al inicio del proyecto lo estaba realizando mediante coordenadas, añadiendo punto por punto, pero era más cansado y a veces terminaba confundíendome, por lo que opte por la matriz.



- $x1$ es el borde derecho: simplemente $x0 + cellW$.
- $y0$ es el borde de abajo del cuadrito. Importante: en nuestro patrón la fila 0 está ARRIBA, pero en OpenGL el +Y está arriba, así que restamos para “bajar”. Por eso: $1.0f - (r + 1) * cellH$.
- $y1$ es el borde de arriba del cuadrito: $y0 + cellH$.

Lo que tendríamos como resultado los siguientes puntos de los triángulos:

Primer triángulo:

($x0, y0$) abajo-izquierda

($x1, y0$) abajo-derecha

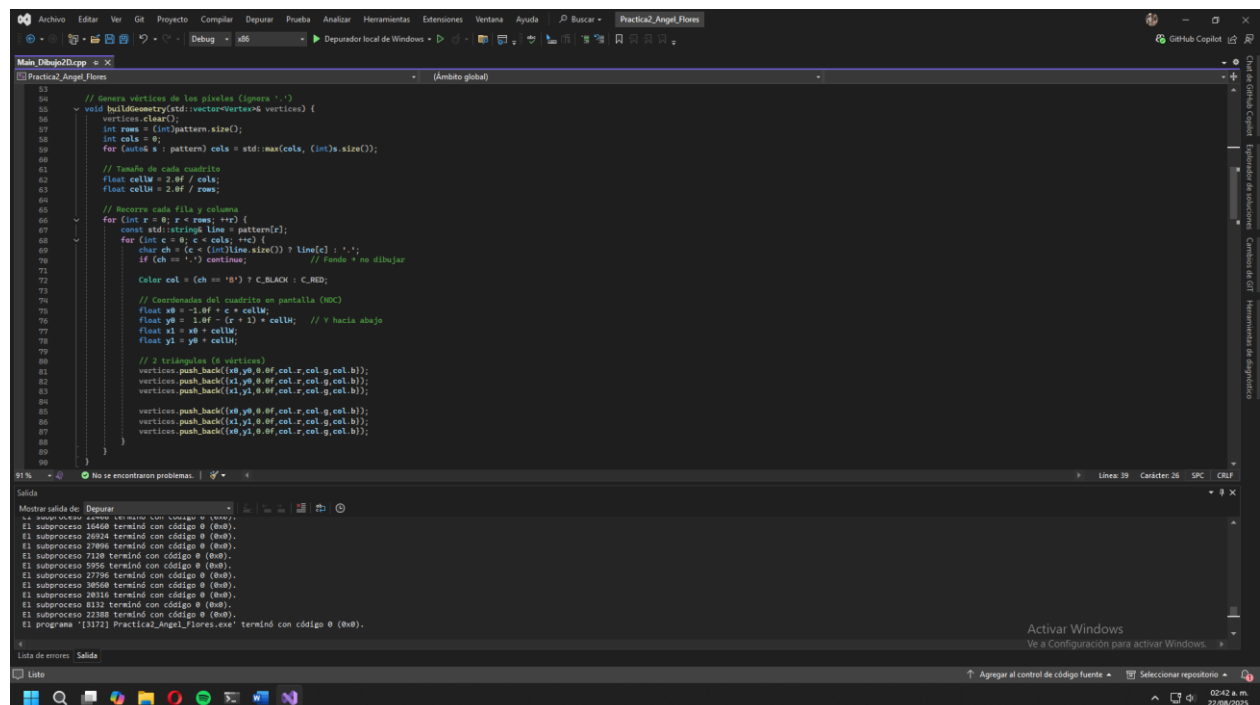
($x1, y1$) arriba-derecha

Segundo triángulo:

($x0, y0$) abajo-izquierda (repetido)

($x1, y1$) arriba-derecha

($x0, y1$) arriba-izquierda



```
53 // Genera vértices de los píxeles (figura 'x')
54 void buildGeometry(std::vector<Vertex*> vertices) {
55     vertices.clear();
56     int rows = (int)pattern.size();
57     int cols = 0;
58     for (const s : pattern) cols = std::max(cols, (int)s.size());
59
60     // Tamaño de cada cuadrado
61     float cellH = 2.0f / rows;
62     float cellW = 2.0f / cols;
63
64     // Recorre cada fila y columna
65     for (int r = 0; r < rows; ++r) {
66         const std::string line = pattern[r];
67         for (int c = 0; c < cols; ++c) {
68             char ch = (c < (int)line.size()) ? line[c] : ' ';
69             if (ch == 'x') continue; // Píxel no dibujar
70
71             Color col = (ch == 'B') ? C_BLACK : C_MED;
72
73             // Coordenadas del cuadrado en pantalla (NDC)
74             float x0 = -1.0f + c * cellW;
75             float y0 = 1.0f - (r + 1) * cellH; // 'y' hacia abajo
76             float x1 = x0 + cellW;
77             float y1 = y0 + cellH;
78
79             // 2 triángulos (6 vértices)
80             vertices.push_back(new Vertex(x0, y0, 0.0f, col.r, col.g, col.b));
81             vertices.push_back(new Vertex(x1, y0, 0.0f, col.r, col.g, col.b));
82             vertices.push_back(new Vertex(x1, y1, 0.0f, col.r, col.g, col.b));
83
84             vertices.push_back(new Vertex(x0, y0, 0.0f, col.r, col.g, col.b));
85             vertices.push_back(new Vertex(x1, y1, 0.0f, col.r, col.g, col.b));
86             vertices.push_back(new Vertex(x0, y1, 0.0f, col.r, col.g, col.b));
87         }
88     }
89 }
```

91% No se encontraron problemas. | 97*

Mostrar salida de Depurador

El proceso 16460 terminó con código 0 (0x0).

El subproceso 26924 terminó con código 0 (0x0).

El subproceso 27966 terminó con código 0 (0x0).

El subproceso 7120 terminó con código 0 (0x0).

El subproceso 2956 terminó con código 0 (0x0).

El subproceso 27796 terminó con código 0 (0x0).

El subproceso 36050 terminó con código 0 (0x0).

El subproceso 28116 terminó con código 0 (0x0).

El subproceso 8132 terminó con código 0 (0x0).

El subproceso 22280 terminó con código 0 (0x0).

El programa [3172] Practica2_Angel_Flores.exe terminó con código 0 (0x0).

Activar Windows
Ve a Configuración para activar Windows.

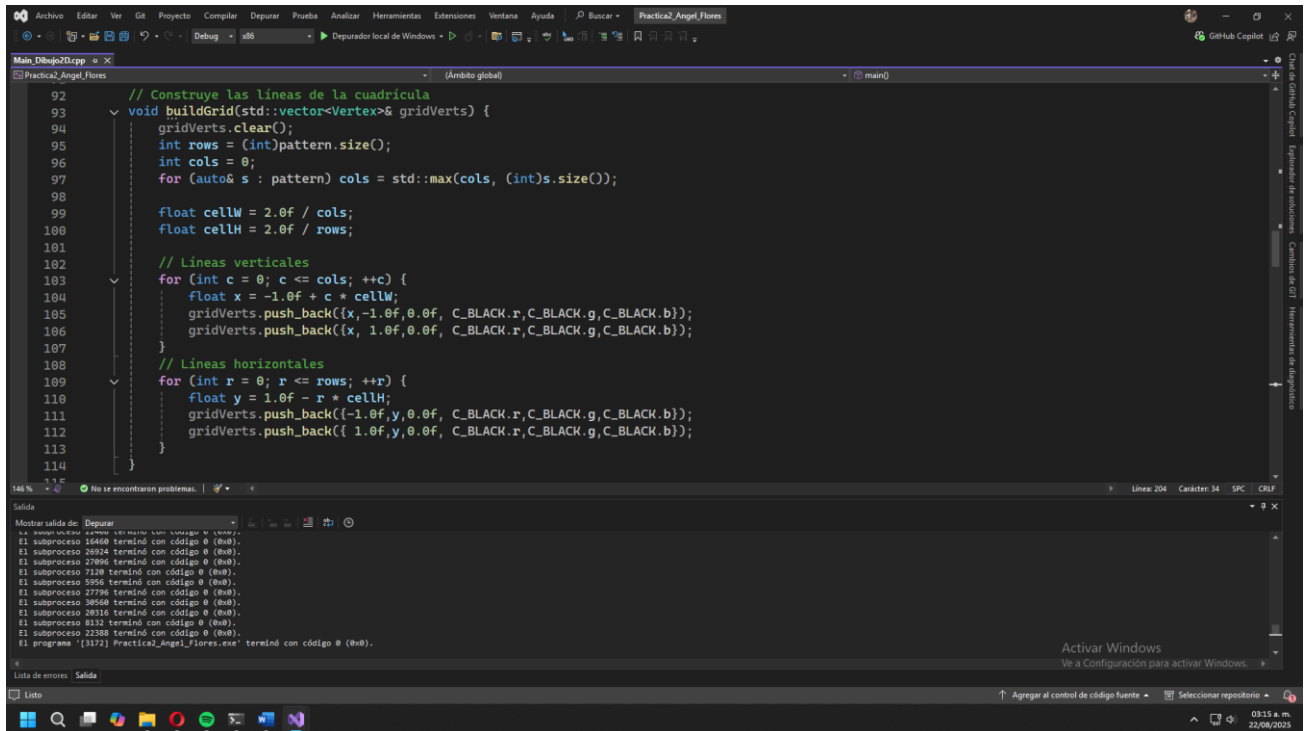
Lista de errores: Salida

Lista

02:42 a.m.
22/08/2025

Una vez que tenemos definidos los vértices de cada píxel yo opté por poner una cuadrícula encima de la imagen para tener una guía (esta puede activarse o desactivarse), para ello lo hice siguiendo una lógica similar a la de la generación de los vértices

Arma una lista de puntos por pares para dibujar las líneas de la cuadrícula que cubre todo el dibujo. Cada línea (vertical u horizontal) son 2 vértices y luego OpenGL las une porque usamos GL_LINES.



```
92 // Construye las líneas de la cuadrícula
93 void buildGrid(std::vector<Vertex>& gridVerts) {
94     gridVerts.clear();
95     int rows = (int)pattern.size();
96     int cols = 0;
97     for (auto& s : pattern) cols = std::max(cols, (int)s.size());
98
99     float cellW = 2.0f / cols;
100    float cellH = 2.0f / rows;
101
102    // Líneas verticales
103    for (int c = 0; c <= cols; ++c) {
104        float x = -1.0f + c * cellW;
105        gridVerts.push_back({x, -1.0f, 0.0f, C_BLACK.r, C_BLACK.g, C_BLACK.b});
106        gridVerts.push_back({x, 1.0f, 0.0f, C_BLACK.r, C_BLACK.g, C_BLACK.b});
107    }
108    // Líneas horizontales
109    for (int r = 0; r <= rows; ++r) {
110        float y = 1.0f - r * cellH;
111        gridVerts.push_back({-1.0f, y, 0.0f, C_BLACK.r, C_BLACK.g, C_BLACK.b});
112        gridVerts.push_back({1.0f, y, 0.0f, C_BLACK.r, C_BLACK.g, C_BLACK.b});
113    }
114 }
```

Salida

Mostrar salida de: Depurar

El proceso 16460 terminó con código 0 (0x0).

El subproceso 26924 terminó con código 0 (0x0).

El subproceso 27806 terminó con código 0 (0x0).

El subproceso 7120 terminó con código 0 (0x0).

El subproceso 5956 terminó con código 0 (0x0).

El subproceso 27796 terminó con código 0 (0x0).

El subproceso 38560 terminó con código 0 (0x0).

El subproceso 28338 terminó con código 0 (0x0).

El subproceso 8132 terminó con código 0 (0x0).

El subproceso 22388 terminó con código 0 (0x0).

El programa '[1372] Practica2_Angel_Flores.exe' terminó con código 0 (0x0).

Finalmente, en el main inicializamos las librerías, preparamos los datos del hongo y la cuadrícula, se sube todo a la GPU, y en un bucle se limpia la pantalla y vuelve a dibujar cada frame hasta que cerremos la ventana.

```

150 // Geometria del hueco
151 std::vector<Vertex> pixels;
152 buildGeometry(pixels);
153
154 // Geometria de la cuadrícula
155 std::vector<Vertex> grid;
156 if (SHOW_GRID) buildGrid(grid);
157
158 // VAO/VBO del sprite
159 GLuint VAO=0, VBO=0;
160 glGenVertexArrays(1,&VAO);
161 glBindVertexArray(VAO);
162 glBufferData(GL_ARRAY_BUFFER, pixels.size()*sizeof(Vertex), pixels.data(), GL_STATIC_DRAW);
163 glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (GLvoid*)0);
164 glEnableVertexAttribArray(0);
165 glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (GLvoid*)(3*sizeof(float)));
166 glBindVertexArray(0);
167
168 // VAO/VBO de la cuadrícula
169 GLuint gridVAO=0, gridVBO=0;
170 if (SHOW_GRID) {
171     glGenVertexArrays(1,&gridVAO);
172     glBindVertexArray(gridVAO);
173     glBufferData(GL_ARRAY_BUFFER, grid.size()*sizeof(Vertex), grid.data(), GL_STATIC_DRAW);
174     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (GLvoid*)0);
175     glEnableVertexAttribArray(0);
176     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (GLvoid*)(3*sizeof(float)));
177     glBindVertexArray(0);
178 }
179
180 int main() {
181     // Inicialización de OpenGL
182     glfwInit();
183     GLFWwindow* window = glfwCreateWindow(800, 800, "Practica2_Angel_Flores", NULL, NULL);
184     if (!window) {
185         std::cerr << "Error al crear la ventana\n";
186         return -1;
187     }
188     glfwMakeContextCurrent(window);
189     glfwSetWindowShouldCloseCallback(window, glfw_should_close);
190     glfwPollEvents();
191
192     // Fondo blanco
193     glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
194     glClear(GL_COLOR_BUFFER_BIT);
195     ourShader.Use();
196
197     // Dibuja sprite
198     glBindVertexArray(VAO);
199     glDrawArrays(GL_TRIANGLES, 0, (GLsizei)pixels.size());
200
201     // Dibuja cuadrícula encima
202     if (SHOW_GRID) {
203         glBindVertexArray(gridVAO);
204         glLineWidth(1.0f);
205         glDrawArrays(GL_LINES, 0, (GLsizei)grid.size());
206     }
207     glBindVertexArray(0);
208     glfwSwapBuffers(window);
209
210     // Limpia memoria GPU
211     glDeleteBuffers(1,&VBO);
212     glDeleteVertexArrays(1,&VAO);
213     if (SHOW_GRID) {
214         glDeleteBuffers(1,&gridVBO);
215         glDeleteVertexArrays(1,&gridVAO);
216     }
217
218     glfwTerminate();
219     return EXIT_SUCCESS;
220 }

```

Salida:

```

E1 subproceso 16460 terminó con código 0 (0x0).
E1 subproceso 28924 terminó con código 0 (0x0).
E1 subproceso 27896 terminó con código 0 (0x0).
E1 subproceso 7120 terminó con código 0 (0x0).
E1 subproceso 5956 terminó con código 0 (0x0).
E1 subproceso 27796 terminó con código 0 (0x0).
E1 subproceso 38560 terminó con código 0 (0x0).
E1 subproceso 28216 terminó con código 0 (0x0).
E1 subproceso 8132 terminó con código 0 (0x0).
E1 subproceso 22388 terminó con código 0 (0x0).
E1 programa '[3172] Practica2_Angel_Flores.exe' terminó con código 0 (0x0).

```

```

180
181
182 // Función principal
183 while (!glfwWindowShouldClose(window)) {
184     glfwPollEvents();
185
186     // Fondo blanco
187     glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
188     glClear(GL_COLOR_BUFFER_BIT);
189     ourShader.Use();
190
191     // Dibuja sprite
192     glBindVertexArray(VAO);
193     glDrawArrays(GL_TRIANGLES, 0, (GLsizei)pixels.size());
194
195     // Dibuja cuadrícula encima
196     if (SHOW_GRID) {
197         glBindVertexArray(gridVAO);
198         glLineWidth(1.0f);
199         glDrawArrays(GL_LINES, 0, (GLsizei)grid.size());
200     }
201     glBindVertexArray(0);
202     glfwSwapBuffers(window);
203
204     // Limpia memoria GPU
205     glDeleteBuffers(1,&VBO);
206     glDeleteVertexArrays(1,&VAO);
207     if (SHOW_GRID) {
208         glDeleteBuffers(1,&gridVBO);
209         glDeleteVertexArrays(1,&gridVAO);
210     }
211
212     glfwTerminate();
213     return EXIT_SUCCESS;
214 }

```

Salida:

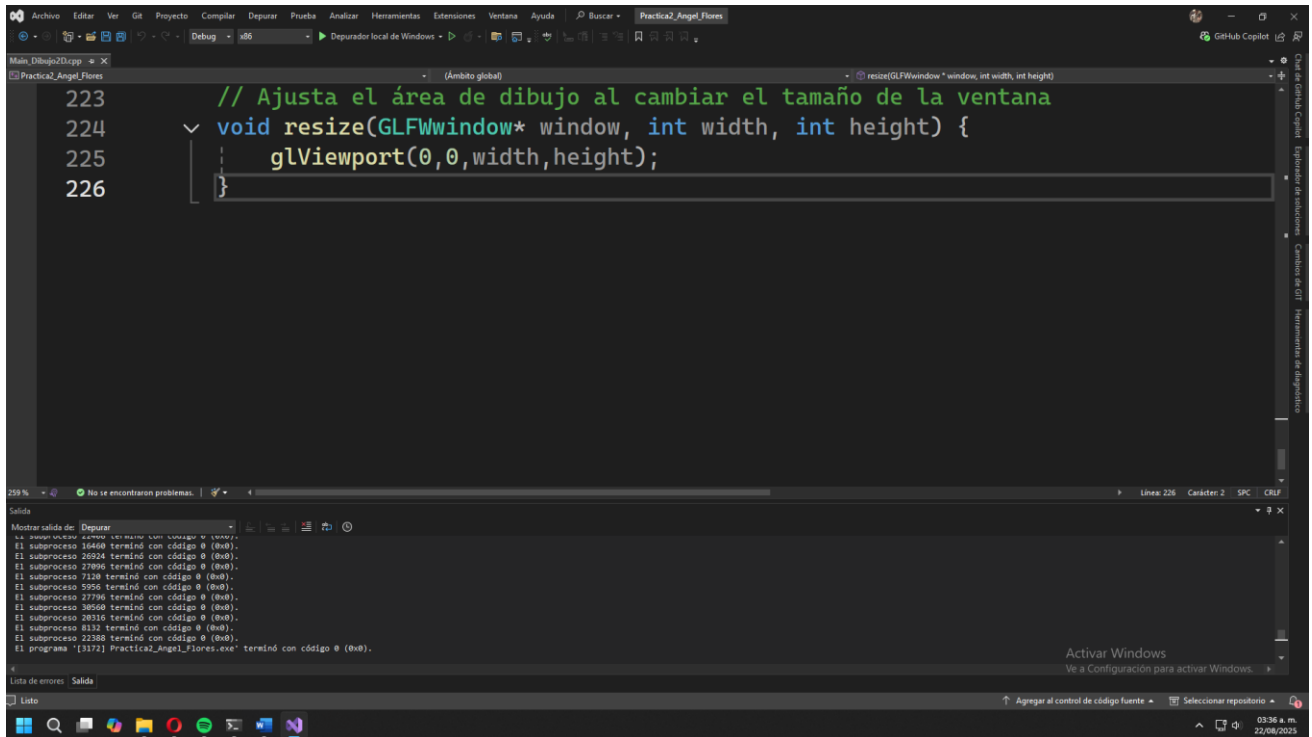
```

E1 subproceso 16460 terminó con código 0 (0x0).
E1 subproceso 28924 terminó con código 0 (0x0).
E1 subproceso 27896 terminó con código 0 (0x0).
E1 subproceso 7120 terminó con código 0 (0x0).
E1 subproceso 5956 terminó con código 0 (0x0).
E1 subproceso 27796 terminó con código 0 (0x0).
E1 subproceso 38560 terminó con código 0 (0x0).
E1 subproceso 28216 terminó con código 0 (0x0).
E1 subproceso 8132 terminó con código 0 (0x0).
E1 subproceso 22388 terminó con código 0 (0x0).
E1 programa '[3172] Practica2_Angel_Flores.exe' terminó con código 0 (0x0).

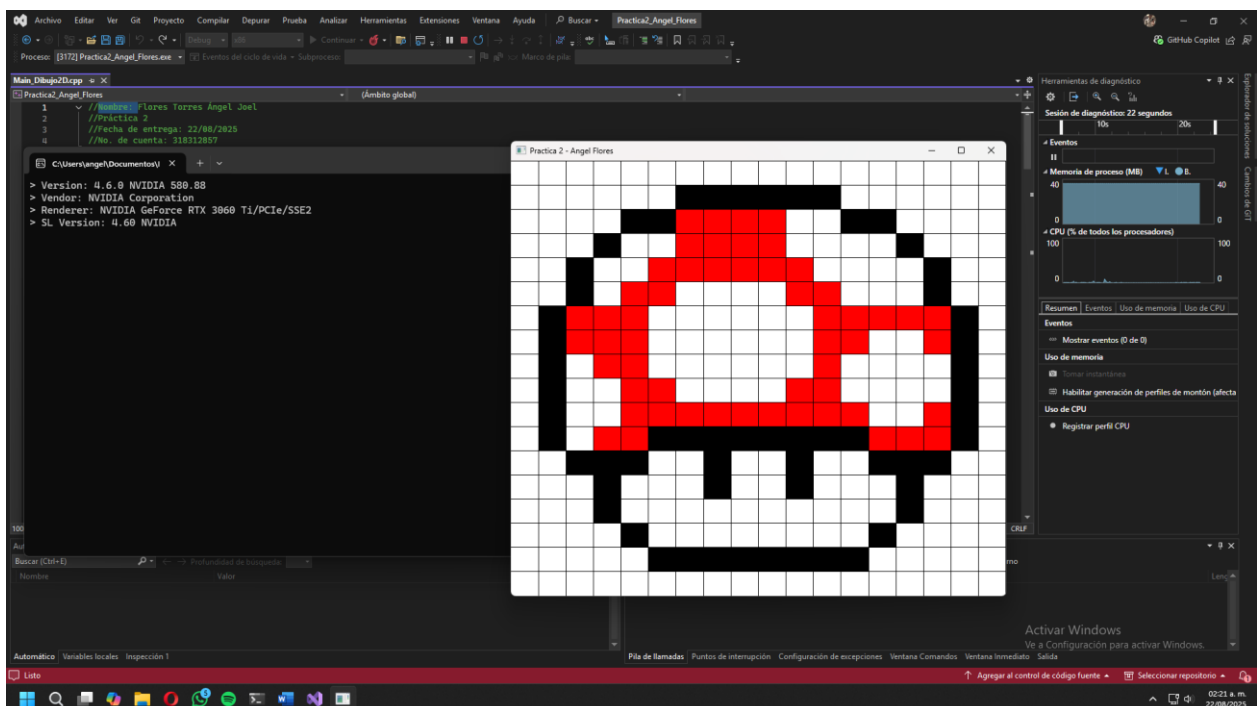
```

Como adición extra añadí la función resize para que al momento de modificar el tamaño de la ventana resultante (donde se muestra el dibujo) OpenGL sepa el nuevo

tamaño disponible y pueda volver a encajar el dibujo dentro de la ventana cuando la redimensionemos (así no se corta la imagen o no queda aplastado)

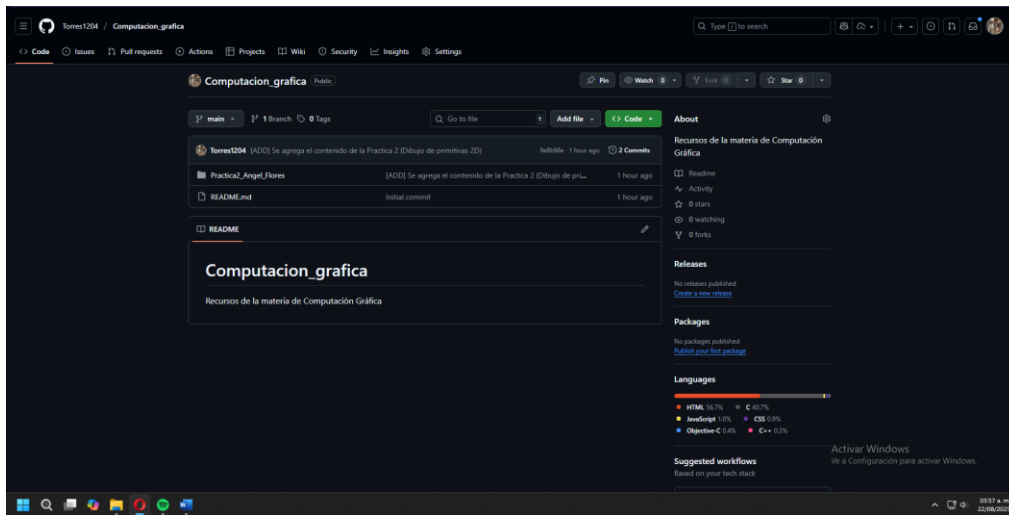


Resultado



Link del repositorio:

https://github.com/Torres1204/Computacion_grafica.git



Referencias

Dynamic Graphics Project, University of Toronto. (s. f.). OpenGL Drawing Primitives. Recuperado el 22 de agosto de 2025, de https://www-dgp-toronto-edu.translate.goog/~ah/csc418/fall_2001/tut/ogl_draw.html?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc

S08 - 1 Primitivas gráficas y algoritmos. (s. f.). Scribd. Recuperado el 22 de agosto de 2025, de <https://es.scribd.com/presentation/381378541/S08-1-Primitivas-Graficas-y-Algoritmos>

Universidad de Chile. (s. f.). Computación gráfica: conceptos fundamentales (Parte II) [PDF]. U-Cursos. Recuperado el 22 de agosto de 2025, de https://www.u-cursos.cl/ingenieria/2012/1/CC3501/2/material_docente/bajar?id_material=420057

Tecnologías Interactivas y Computación Gráfica. (s. f.). Dibujo de primitivas en 2D con shaders en OpenGL [Video]. YouTube. Recuperado el 22 de agosto de 2025, de <https://youtu.be/UWcnvhSDYYA?si=3A6JRf2wW4yYr1LE>