public class ChunkLoaderThread extends Thread {

/\*\*

\* Cola de coordenadas (chunkX, chunkZ) pendientes de cargar/generar

\*/

private final Queue<Vector2i> pending = new ConcurrentLinkedQueue<>();

private final ConcurrentHashMap<Vector2i, Boolean> pendingMap = new ConcurrentHashMap<>();

private int count = 1;

/\*\*

\* Referencia al WorldManager, para delegar carga/generación y encolar el

\* chunk

\*/

private final WorldManager worldManager;

/\*\*

\* Bandera para detener el hilo de forma limpia

\*/

private volatile boolean running = true;

public ChunkLoaderThread(WorldManager worldManager) {

this.worldManager = worldManager;

setName("ChunkLoaderThread");

setDaemon(true);

}

/\*\*

\* Solicita la carga o generación de un chunk (solo lo encola). No vuelve a

\* encolar la misma coordenada si ya está pendiente.

\*/

public void requestLoad(int chunkX, int chunkZ) {

Vector2i key = new Vector2i(chunkX, chunkZ);

if (!pendingMap.containsKey(key) && !isAlreadyLoaded(chunkX, chunkZ)) {

pending.add(key);

pendingMap.put(key, true);

}

}

public boolean isAlreadyLoaded(int ChunkX, int ChunkZ) {

return worldManager.getChunks().containsKey(ChunkX + ",0," + ChunkZ);

}

/\*\*

\* Marca el hilo para que deje de ejecutarse y lo interrumpe si está dormido

\*/

public void terminate() {

running = false;

this.interrupt();

}

public boolean hasPendingRequest(int chunkX, int chunkZ) {

return pending.contains(new Vector2i(chunkX, chunkZ));

}

@Override

public void run() {

while (running) {

int processed = 0;

// Procesamos hasta 3 chunks por iteración (prioridad a los más cercanos)

while (processed < 3) {

Vector2i pos = pending.poll();

if (pos == null) {

break; // No hay más peticiones en esta ronda

}

String key = pos.x + ",0," + pos.y;

if (worldManager.getChunks().containsKey(key)) {

// Ya está cargado, lo saltamos

pendingMap.remove(pos);

continue;

}

// Si no está cargado, lo generamos

Chunk chunk = new Chunk(pos.x, 0, pos.y);

boolean loadedFromDisk = worldManager.loadChunk(chunk);

if (!loadedFromDisk) {

chunk.generate();

}

worldManager.enqueueChunkToMesh(chunk);

pendingMap.remove(pos);

pendingMap.remove(pos);

processed++;

}

// Dormimos un breve rato para no saturar la CPU

try {

Thread.sleep(10);

} catch (InterruptedException ignored) {

// Si nos interrumpen, revisamos la bandera 'running' en la siguiente vuelta

}

}

}

}

public class ChunkSaveThread extends Thread {

private final WorldManager worldManager;

// Cola concurrente de chunks pendientes de guardado

private final Set<Chunk> pendingSaves = ConcurrentHashMap.newKeySet();

private volatile boolean running = true;

public ChunkSaveThread(WorldManager worldManager) {

this.worldManager = worldManager;

setName("ChunkSaveThread");

setDaemon(true); // para que no impida cerrar la app

}

/\*\*

\* Encolar un chunk para guardarlo en disco. Si ya está en la cola, no se

\* vuelve a encolar para evitar duplicados.

\*/

public void requestSave(Chunk chunk) {

pendingSaves.add(chunk);

}

@Override

public void run() {

while (running) {

Chunk chunk = null;

Iterator<Chunk> it = pendingSaves.iterator();

if (it.hasNext()) {

chunk = it.next();

it.remove(); // Importante: eliminar antes de guardar

}

if (chunk != null) {

try {

worldManager.saveChunkImmediate(chunk);

} catch (IOException ex) {

System.err.println("Error al guardar el chunk " + chunk.getKey() + ": " + ex.getMessage());

ex.printStackTrace();

}

} else {

// Si no hay chunks por guardar, duerme un poco

try {

Thread.sleep(50); // Reduce uso de CPU

} catch (InterruptedException ignored) {

}

}

}

}

public void terminate() {

running = false;

this.interrupt();

}

}

public class Cube {

private final Mesh mesh;

private Vector3f rotation = new Vector3f();

private Texture texture;

private float[] vertices;

public Cube() throws Exception {

vertices = new float[]{

// Cara frontal (Columna 1, Fila 0)

-0.5f, 0.5f, 0.5f, 0f, -1f, 0f, 0.33f, 0.00f,1.0f,1.0f,

-0.5f, -0.5f, 0.5f, 0f, -1f, 0f, 0.33f, 0.50f,1.0f,1.0f,

0.5f, -0.5f, 0.5f, 0f, -1f, 0f, 0.66f, 0.50f,1.0f,1.0f,

0.5f, -0.5f, 0.5f, 0f, -1f, 0f, 0.66f, 0.50f,1.0f,1.0f,

0.5f, 0.5f, 0.5f, 0f, -1f, 0f, 0.66f, 0.00f,1.0f,1.0f,

-0.5f, 0.5f, 0.5f, 0f, -1f, 0f, 0.33f, 0.00f,1.0f,1.0f,

// Cara trasera (Columna 1, Fila 1)

0.5f, 0.5f, -0.5f, 0f, 0f, -1f, 0.66f, 0.50f,1.0f,1.0f,

0.5f, -0.5f, -0.5f, 0f, 0f, -1f, 0.66f, 1.00f,1.0f,1.0f,

-0.5f, -0.5f, -0.5f, 0f, 0f, -1f, 0.33f, 1.00f,1.0f,1.0f,

-0.5f, -0.5f, -0.5f, 0f, 0f, -1f, 0.33f, 1.00f,1.0f,1.0f,

-0.5f, 0.5f, -0.5f, 0f, 0f, -1f, 0.33f, 0.50f,1.0f,1.0f,

0.5f, 0.5f, -0.5f, 0f, 0f, -1f, 0.66f, 0.50f,1.0f,1.0f,

// Cara izquierda (Columna 0, Fila 1)

-0.5f, 0.5f, -0.5f, -1f, 0f, 0f, 0.00f, 0.50f,1.0f,1.0f,

-0.5f, -0.5f, -0.5f, -1f, 0f, 0f, 0.00f, 1.00f,1.0f,1.0f,

-0.5f, -0.5f, 0.5f, -1f, 0f, 0f, 0.33f, 1.00f,1.0f,1.0f,

-0.5f, -0.5f, 0.5f, -1f, 0f, 0f, 0.33f, 1.00f,1.0f,1.0f,

-0.5f, 0.5f, 0.5f, -1f, 0f, 0f, 0.33f, 0.50f,1.0f,1.0f,

-0.5f, 0.5f, -0.5f, -1f, 0f, 0f, 0.00f, 0.50f,1.0f,1.0f,

// Cara derecha (Columna 2, Fila 1)

0.5f, 0.5f, 0.5f, 1f, 0f, 0f, 0.66f, 0.50f,1.0f,1.0f,

0.5f, -0.5f, 0.5f, 1f, 0f, 0f, 0.66f, 1.00f,1.0f,1.0f,

0.5f, -0.5f, -0.5f, 1f, 0f, 0f, 1.00f, 1.00f,1.0f,1.0f,

0.5f, -0.5f, -0.5f, 1f, 0f, 0f, 1.00f, 1.00f,1.0f,1.0f,

0.5f, 0.5f, -0.5f, 1f, 0f, 0f, 1.00f, 0.50f,1.0f,1.0f,

0.5f, 0.5f, 0.5f, 1f, 0f, 0f, 0.66f, 0.50f,1.0f,1.0f,

// Cara superior (Columna 0, Fila 0)

-0.5f, 0.5f, -0.5f, 0f, 1f, 0f, 0.00f, 0.00f,1.0f,1.0f,

-0.5f, 0.5f, 0.5f, 0f, 1f, 0f, 0.00f, 0.50f,1.0f,1.0f,

0.5f, 0.5f, 0.5f, 0f, 1f, 0f, 0.33f, 0.50f,1.0f,1.0f,

0.5f, 0.5f, 0.5f, 0f, 1f, 0f, 0.33f, 0.50f,1.0f,1.0f,

0.5f, 0.5f, -0.5f, 0f, 1f, 0f, 0.33f, 0.00f,1.0f,1.0f,

-0.5f, 0.5f, -0.5f, 0f, 1f, 0f, 0.00f, 0.00f,1.0f,1.0f,

// Cara inferior (Columna 2, Fila 0)

-0.5f, -0.5f, 0.5f, 0f, -1f, 0f, 0.66f, 0.50f,1.0f,1.0f,

-0.5f, -0.5f, -0.5f, 0f, -1f, 0f, 0.66f, 0.00f,1.0f,1.0f,

0.5f, -0.5f, -0.5f, 0f, -1f, 0f, 1.00f, 0.00f,1.0f,1.0f,

0.5f, -0.5f, -0.5f, 0f, -1f, 0f, 1.00f, 0.00f,1.0f,1.0f,

0.5f, -0.5f, 0.5f, 0f, -1f, 0f, 1.00f, 0.50f,1.0f,1.0f,

-0.5f, -0.5f, 0.5f, 0f, -1f, 0f, 0.66f, 0.50f,1.0f,1.0f

};

mesh = new Mesh(vertices);

}

public void setRotation(Vector3f rotation) {

this.rotation.set(rotation);

}

public void render(Texture texture, Matrix4f projection, Matrix4f view, ShaderProgram shader) {

this.texture = texture;

shader.bind();

Matrix4f model = new Matrix4f()

.translate(-1.7f, 0.7f, 0f)

.scale(0.1f)

.rotateLocalY(0.35f)

.rotateX((float) Math.toRadians(rotation.x))

.rotateY((float) Math.toRadians(rotation.y))

.rotateZ((float) Math.toRadians(rotation.z));

shader.setUniformMat4("model", model);

shader.setUniformMat4("view", view);

shader.setUniformMat4("projection", projection);

glActiveTexture(GL\_TEXTURE0);

this.texture.bind();

shader.setUniform1i("textureSampler", 0);

mesh.render();

shader.unbind();

}

public float[] getVErt() {

return vertices;

}

public void cleanup() {

mesh.cleanup();

}

}

public class CubeFace {

// Vértices por cara, sentido horario desde esquina inferior izquierda

public static float[][] getFaceVertices(Direction dir) {

switch (dir) {

case UP:

return new float[][] {

{0, 1, 1}, {1, 1, 1}, {1, 1, 0}, {0, 1, 0}

};

case DOWN:

return new float[][] {

{0, 0, 0}, {1, 0, 0}, {1, 0, 1}, {0, 0, 1}

};

case FRONT:

return new float[][] {

{0, 0, 1}, {1, 0, 1}, {1, 1, 1}, {0, 1, 1}

};

case BACK:

return new float[][] {

{1, 0, 0}, {0, 0, 0}, {0, 1, 0}, {1, 1, 0}

};

case LEFT:

return new float[][] {

{0, 0, 0}, {0, 0, 1}, {0, 1, 1}, {0, 1, 0}

};

case RIGHT:

return new float[][] {

{1, 0, 1}, {1, 0, 0}, {1, 1, 0}, {1, 1, 1}

};

}

return new float[0][0];

}

public static float[] getNormal(Direction dir) {

switch (dir) {

case UP: return new float[] {0, 1, 0};

case DOWN: return new float[] {0, -1, 0};

case FRONT: return new float[] {0, 0, 1};

case BACK: return new float[] {0, 0, -1};

case LEFT: return new float[] {-1, 0, 0};

case RIGHT: return new float[] {1, 0, 0};

}

return new float[] {0, 0, 0};

}

}

public class CubeGenerator {

// Tamaño de cada celda de la textura (col: 3 columnas, row: 2 filas)

private static final float UV\_WIDTH = 1.0f / 3.0f;

private static final float UV\_HEIGHT = 1.0f / 2.0f;

public enum Direction {

UP(0, +1, 0),

DOWN(0, -1, 0),

FRONT(0, 0, +1),

BACK(0, 0, -1),

LEFT(-1, 0, 0),

RIGHT(+1, 0, 0);

public final int offsetX, offsetY, offsetZ;

Direction(int dx, int dy, int dz) {

this.offsetX = dx;

this.offsetY = dy;

this.offsetZ = dz;

}

}

public static float[] createCube(int cx, int cy, int cz, Block[][][] blocks) {

List<Float> data = new ArrayList<>(6 \* 10 \* 6);

//para cada cara

for (Direction dir : Direction.values()) {

int texCol = getTexCol(dir);

int texRow = getTexRow(dir);

float[][] positions = CubeFace.getFaceVertices(dir);

float[] normal = CubeFace.getNormal(dir);

float u0 = texCol \* UV\_WIDTH, v0 = texRow \* UV\_HEIGHT;

float u1 = u0 + UV\_WIDTH, v1 = v0 + UV\_HEIGHT;

// UVs: se asignan en sentido horario desde esquina inferior izquierda

float[][] uvs = new float[][]{

{u0, v1},

{u1, v1},

{u1, v0},

{u0, v0}

};

int[][] triIndices = new int[][]{

{0, 1, 2},

{2, 3, 0}

};

for (int[] tri : triIndices) {

for (int vi : tri) {

float px = positions[vi][0] + cx;

float py = positions[vi][1] + cy;

float pz = positions[vi][2] + cz;

int nx = (int) positions[vi][0];

int ny = (int) positions[vi][1];

int nz = (int) positions[vi][2];

float shadow = calcShadow(dir, cy);

//AO

int ix = (int) Math.floor(px);

int iy = (int) Math.floor(py);

int iz = (int) Math.floor(pz);

float ao = calAO(nx, ny, nz, dir, blocks, cx, cy, cz);

//UV

float u = uvs[vi][0];

float v = uvs[vi][1];

data.add(px);

data.add(py);

data.add(pz);

data.add(normal[0]);

data.add(normal[1]);

data.add(normal[2]);

data.add(u);

data.add(v);

data.add(shadow);

data.add(ao);

}

}

}

float[] result = new float[data.size()];

for (int i = 0; i < data.size(); i++) {

result[i] = data.get(i);

}

return result;

}

private static float calcShadow(Direction dir, int cy) {

if (cy == 0) {

return 0.1f;

}

if (cy == 1) {

return 0.2f;

}

if (cy == 2) {

return 0.4f;

}

if (cy == 3) {

return 0.6f;

}

if (cy == 4) {

if (dir.equals(dir.UP)) {

return 1.0f;

}

return 0.8f;

}

return 1.0f;

}

private static float calAO(int x, int y, int z, Direction dir, Block[][][] blocks, int ix, int iy, int iz) {

boolean lado1 = false, lado2 = false, esquina = false;

String vertex = x + "," + y + "," + z;

//System.out.println("Es el vertice: ("+vertex+") de la cara "+ dir.toString() );

switch (dir) {

case FRONT:

switch (vertex) {

case "0,0,1":

lado1 = isSolidBlock(blocks, ix - 1, iy, iz + 1);

lado2 = isSolidBlock(blocks, ix, iy - 1, iz + 1);

esquina = isSolidBlock(blocks, ix - 1, iy - 1, iz + 1);

break;

case "1,0,1":

lado1 = isSolidBlock(blocks, ix + 1, iy, iz + 1);

lado2 = isSolidBlock(blocks, ix, iy - 1, iz + 1);

esquina = isSolidBlock(blocks, ix + 1, iy - 1, iz + 1);

break;

case "1,1,1":

lado1 = isSolidBlock(blocks, ix + 1, iy, iz + 1);

lado2 = isSolidBlock(blocks, ix, iy + 1, iz + 1);

esquina = isSolidBlock(blocks, ix + 1, iy + 1, iz + 1);

break;

case "0,1,1":

lado1 = isSolidBlock(blocks, ix - 1, iy, iz + 1);

lado2 = isSolidBlock(blocks, ix, iy + 1, iz + 1);

esquina = isSolidBlock(blocks, ix - 1, iy + 1, iz + 1);

break;

}

break;

case BACK:

switch (vertex) {

case "1,0,0":

lado1 = isSolidBlock(blocks, ix + 1, iy, iz - 1);

lado2 = isSolidBlock(blocks, ix, iy - 1, iz - 1);

esquina = isSolidBlock(blocks, ix + 1, iy - 1, iz - 1);

break;

case "0,0,0":

lado1 = isSolidBlock(blocks, ix - 1, iy, iz - 1);

lado2 = isSolidBlock(blocks, ix, iy - 1, iz - 1);

esquina = isSolidBlock(blocks, ix - 1, iy - 1, iz - 1);

break;

case "0,1,0":

lado1 = isSolidBlock(blocks, ix - 1, iy, iz - 1);

lado2 = isSolidBlock(blocks, ix, iy + 1, iz + -1);

esquina = isSolidBlock(blocks, ix - 1, iy + 1, iz - 1);

break;

case "1,1,0":

lado1 = isSolidBlock(blocks, ix + 1, iy, iz - 1);

lado2 = isSolidBlock(blocks, ix, iy + 1, iz - 1);

esquina = isSolidBlock(blocks, ix + 1, iy + 1, iz - 1);

break;

}

break;

case LEFT:

switch (vertex) {

case "0,0,0":

lado1 = isSolidBlock(blocks, ix - 1, iy, iz - 1);

lado2 = isSolidBlock(blocks, ix - 1, iy - 1, iz);

esquina = isSolidBlock(blocks, ix - 1, iy - 1, iz - 1);

break;

case "0,0,1":

lado1 = isSolidBlock(blocks, ix - 1, iy, iz + 1);

lado2 = isSolidBlock(blocks, ix - 1, iy - 1, iz);

esquina = isSolidBlock(blocks, ix - 1, iy - 1, iz + 1);

break;

case "0,1,1":

lado1 = isSolidBlock(blocks, ix - 1, iy, iz + 1);

lado2 = isSolidBlock(blocks, ix - 1, iy + 1, iz);

esquina = isSolidBlock(blocks, ix - 1, iy + 1, iz + 1);

break;

case "0,1,0":

lado1 = isSolidBlock(blocks, ix - 1, iy, iz - 1);

lado2 = isSolidBlock(blocks, ix - 1, iy + 1, iz);

esquina = isSolidBlock(blocks, ix - 1, iy + 1, iz - 1);

break;

}

break;

case RIGHT:

switch (vertex) {

case "1,0,1":

lado1 = isSolidBlock(blocks, ix + 1, iy, iz + 1);

lado2 = isSolidBlock(blocks, ix + 1, iy - 1, iz);

esquina = isSolidBlock(blocks, ix + 1, iy - 1, iz + 1);

break;

case "1,0,0":

lado1 = isSolidBlock(blocks, ix + 1, iy, iz - 1);

lado2 = isSolidBlock(blocks, ix + 1, iy - 1, iz);

esquina = isSolidBlock(blocks, ix + 1, iy - 1, iz - 1);

break;

case "1,1,0":

lado1 = isSolidBlock(blocks, ix+1, iy, iz - 1);

lado2 = isSolidBlock(blocks, ix+1, iy + 1, iz);

esquina = isSolidBlock(blocks, ix+1, iy + 1, iz - 1);

break;

case "1,1,1":

lado1 = isSolidBlock(blocks, ix + 1, iy, iz + 1);

lado2 = isSolidBlock(blocks, ix + 1, iy + 1, iz);

esquina = isSolidBlock(blocks, ix + 1, iy + 1, iz +1);

break;

}

break;

case UP:

switch (vertex) {

case "0,1,1":

lado1 = isSolidBlock(blocks, ix - 1, iy + 1, iz);

lado2 = isSolidBlock(blocks, ix, iy + 1, iz + 1);

esquina = isSolidBlock(blocks, ix - 1, iy + 1, iz + 1);

break;

case "1,1,1":

lado1 = isSolidBlock(blocks, ix + 1, iy + 1, iz);

lado2 = isSolidBlock(blocks, ix, iy + 1, iz + 1);

esquina = isSolidBlock(blocks, ix + 1, iy + 1, iz + 1);

break;

case "1,1,0":

lado1 = isSolidBlock(blocks, ix + 1, iy + 1, iz);

lado2 = isSolidBlock(blocks, ix, iy + 1, iz - 1);

esquina = isSolidBlock(blocks, ix + 1, iy + 1, iz - 1);

break;

case "0,1,0":

lado1 = isSolidBlock(blocks, ix - 1, iy + 1, iz);

lado2 = isSolidBlock(blocks, ix, iy + 1, iz - 1);

esquina = isSolidBlock(blocks, ix - 1, iy + 1, iz - 1);

break;

}

break;

case DOWN:

switch (vertex) {

case "0,0,0":

lado1 = isSolidBlock(blocks, ix - 1, iy - 1, iz);

lado2 = isSolidBlock(blocks, ix, iy - 1, iz - 1);

esquina = isSolidBlock(blocks, ix - 1, iy - 1, iz - 1);

break;

case "1,0,0":

lado1 = isSolidBlock(blocks, ix + 1, iy - 1, iz);

lado2 = isSolidBlock(blocks, ix, iy - 1, iz - 1);

esquina = isSolidBlock(blocks, ix + 1, iy - 1, iz - 1);

break;

case "1,0,1":

lado1 = isSolidBlock(blocks, ix + 1, iy - 1, iz);

lado2 = isSolidBlock(blocks, ix, iy - 1, iz + 1);

esquina = isSolidBlock(blocks, ix + 1, iy - 1, iz + 1);

break;

case "0,0,1":

lado1 = isSolidBlock(blocks, ix - 1, iy - 1, iz);

lado2 = isSolidBlock(blocks, ix, iy - 1, iz + 1);

esquina = isSolidBlock(blocks, ix - 1, iy - 1, iz + 1);

break;

}

break;

}

//ambos lados = sombra total

if (lado1 && lado2) {

return 0.2f;

}

//contamos cuantos de los 3 estan solidos

int totalSolid = 0;

if (lado1) {

totalSolid++;

}

if (lado2) {

totalSolid++;

}

if (esquina) {

totalSolid++;

}

return (3 - totalSolid) / 3.0f;

}

private static boolean isAir(Block[][][] blocks, int x, int y, int z) {

if (x < 0 || x >= blocks.length

|| y < 0 || y >= blocks[0].length

|| z < 0 || z >= blocks[0][0].length) {

return true; // Consideramos aire fuera de los límites

}

if (blocks[x][y][z].isSolid && blocks[x][y][z] != null) {

return false;

}

return true;

}

private static boolean isInShadow(int x, int y, int z, Direction dir) {

final int MAX\_DIST = 8;

for (int i = 1; i <= MAX\_DIST; i++) {

int nx = x + dir.offsetX \* i;

int ny = y + dir.offsetY \* i;

int nz = z + dir.offsetZ \* i;

Block b = WorldManager.instance.getBlockIfLoader(nx, ny, nz);

if (b == null) {

continue;

}

if (b.isSolid()) {

return true;

}

}

return false;

}

/\*\*

\* Comprueba si en las coordenadas locales (x,y,z) existe un bloque sólido.

\* Debemos verificar límites del chunk:

\*/

private static boolean isSolidBlock(Block[][][] blocks, int lx, int ly, int lz) {

if (lx < 0 || lx >= blocks.length

|| ly < 0 || ly >= blocks[0].length

|| lz < 0 || lz >= blocks[0][0].length) {

return false; // fuera de este chunk → considerar “vacío” para AO

}

Block b = blocks[lx][ly][lz];

return (b != null && b.isSolid());

}

private static int getTexCol(Direction dir) {

switch (dir) {

case UP:

return 0;

case DOWN:

return 2;

case FRONT:

return 1;

case BACK:

return 1;

case LEFT:

return 0;

case RIGHT:

return 2;

}

return 0;

}

private static int getTexRow(Direction dir) {

switch (dir) {

case UP:

return 0;

case DOWN:

return 0;

case FRONT:

return 0;

case BACK:

return 1;

case LEFT:

return 1;

case RIGHT:

return 1;

}

return 0;

}

}

public class CubeSelected {

private int vaoId;

private int vboId;

private int vertexCount;

public CubeSelected() {

float[] vertices = {

// Posiciones // Normales // UVs

// Cara frontal

0, 0, 1, 0, 0, 1, 0, 1,

1, 0, 1, 0, 0, 1, 1, 1,

1, 1, 1, 0, 0, 1, 1, 0,

1, 1, 1, 0, 0, 1, 1, 0,

0, 1, 1, 0, 0, 1, 0, 0,

0, 0, 1, 0, 0, 1, 0, 1,

// Cara trasera

1, 0, 0, 0, 0, -1, 0, 1,

0, 0, 0, 0, 0, -1, 1, 1,

0, 1, 0, 0, 0, -1, 1, 0,

0, 1, 0, 0, 0, -1, 1, 0,

1, 1, 0, 0, 0, -1, 0, 0,

1, 0, 0, 0, 0, -1, 0, 1,

// Cara izquierda

0, 0, 0, -1, 0, 0, 0, 1,

0, 0, 1, -1, 0, 0, 1, 1,

0, 1, 1, -1, 0, 0, 1, 0,

0, 1, 1, -1, 0, 0, 1, 0,

0, 1, 0, -1, 0, 0, 0, 0,

0, 0, 0, -1, 0, 0, 0, 1,

// Cara derecha

1, 0, 1, 1, 0, 0, 0, 1,

1, 0, 0, 1, 0, 0, 1, 1,

1, 1, 0, 1, 0, 0, 1, 0,

1, 1, 0, 1, 0, 0, 1, 0,

1, 1, 1, 1, 0, 0, 0, 0,

1, 0, 1, 1, 0, 0, 0, 1,

// Cara superior

0, 1, 1, 0, 1, 0, 0, 1,

1, 1, 1, 0, 1, 0, 1, 1,

1, 1, 0, 0, 1, 0, 1, 0,

1, 1, 0, 0, 1, 0, 1, 0,

0, 1, 0, 0, 1, 0, 0, 0,

0, 1, 1, 0, 1, 0, 0, 1,

// Cara inferior

0, 0, 0, 0, -1, 0, 0, 1,

1, 0, 0, 0, -1, 0, 1, 1,

1, 0, 1, 0, -1, 0, 1, 0,

1, 0, 1, 0, -1, 0, 1, 0,

0, 0, 1, 0, -1, 0, 0, 0,

0, 0, 0, 0, -1, 0, 0, 1

};

vertexCount = vertices.length / 8;

vaoId = glGenVertexArrays();

glBindVertexArray(vaoId);

vboId = glGenBuffers();

glBindBuffer(GL\_ARRAY\_BUFFER, vboId);

glBufferData(GL\_ARRAY\_BUFFER, vertices, GL\_STATIC\_DRAW);

// Posición (vec3)

glVertexAttribPointer(0, 3, GL\_FLOAT, false, 8 \* Float.BYTES, 0);

glEnableVertexAttribArray(0);

// Normal (vec3)

glVertexAttribPointer(1, 3, GL\_FLOAT, false, 8 \* Float.BYTES, 3 \* Float.BYTES);

glEnableVertexAttribArray(1);

// UV (vec2)

glVertexAttribPointer(2, 2, GL\_FLOAT, false, 8 \* Float.BYTES, 6 \* Float.BYTES);

glEnableVertexAttribArray(2);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glBindVertexArray(0);

}

public void render() {

glBindVertexArray(vaoId);

glDrawArrays(GL\_TRIANGLES, 0, vertexCount);

glBindVertexArray(0);

}

public void cleanup() {

glDeleteBuffers(vboId);

glDeleteVertexArrays(vaoId);

}

}

public class FloatArray {

private float[] data;

private int size;

public FloatArray() {

data = new float[1024];

size = 0;

}

public void add(float[] values) {

ensureCapacity(size + values.length);

for (float v : values) {

data[size++] = v;

}

}

public float[] toArray() {

return Arrays.copyOf(data, size);

}

private void ensureCapacity(int minCapacity) {

if (minCapacity > data.length) {

int newCapacity = Math.max(minCapacity, data.length \* 2);

data = Arrays.copyOf(data, newCapacity);

}

}

}

public class MeshBorder {

private final int vaoId;

private final int vboId;

private final int vertexCount;

public MeshBorder(float[] positions) {

vertexCount = positions.length / 3;

vaoId = glGenVertexArrays();

glBindVertexArray(vaoId);

vboId = glGenBuffers();

glBindBuffer(GL\_ARRAY\_BUFFER, vboId);

FloatBuffer buffer = BufferUtils.createFloatBuffer(positions.length);

buffer.put(positions).flip();

glBufferData(GL\_ARRAY\_BUFFER, buffer, GL\_STATIC\_DRAW);

glVertexAttribPointer(0, 3, GL\_FLOAT, false, 3 \* Float.BYTES, 0);

glEnableVertexAttribArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glBindVertexArray(0);

}

public int getVaoId() {

return vaoId;

}

public int getVboId() {

return vboId;

}

public void render() {

glBindVertexArray(vaoId);

glEnableVertexAttribArray(0);

glDrawArrays(GL\_LINES, 0, vertexCount);

glDisableVertexAttribArray(0);

glBindVertexArray(0);

}

public void cleanup() {

glDisableVertexAttribArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glDeleteBuffers(vboId);

glBindVertexArray(0);

glDeleteVertexArrays(vaoId);

}

}

public class MeshBuilderThread extends Thread {

private final WorldManager worldManager;

private volatile boolean running = true;

public MeshBuilderThread(WorldManager wm) {

this.worldManager = wm;

setName("MeshBuilderThread");

setDaemon(true);

}

public void terminate() {

running = false;

this.interrupt();

}

@Override

public void run() {

while (running) {

Chunk chunk = worldManager.chunksToMesh.poll();

if (chunk != null) {

worldManager.enqueueChunkToAdd(chunk);

}else{

try{Thread.sleep(10);}catch(Exception e){}

}

}

}

}

public class RaycastResult {

public final Vector3f blockPos;

public final Vector3f faceNormal;

public RaycastResult(Vector3f blockPos, Vector3f faceNormal) {

this.blockPos = blockPos;

this.faceNormal = faceNormal;

}

}

public class RaycastUtils {

public static RaycastResult raycast(WorldManager world, Vector3f origin, Vector3f direction, float maxDistance) {

Vector3f ray = new Vector3f(direction).normalize();

Vector3f pos = new Vector3f(origin);

int lastX = (int) Math.floor(pos.x);

int lastY = (int) Math.floor(pos.y);

int lastZ = (int) Math.floor(pos.z);

for (int i = 0; i < maxDistance \* 10; i++) {

pos.add(new Vector3f(ray).mul(0.1f));

int x = (int) Math.floor(pos.x);

int y = (int) Math.floor(pos.y);

int z = (int) Math.floor(pos.z);

if ((x != lastX || y != lastY || z != lastZ)

&& world.getBlockIfLoader(x, y, z) != null

&& world.getBlockIfLoader(x, y, z) != Block.AIR) {

Vector3f blockPos = new Vector3f(x, y, z);

Vector3f faceNormal = new Vector3f(lastX - x, lastY - y, lastZ - z);

return new RaycastResult(blockPos, faceNormal);

}

lastX = x;

lastY = y;

lastZ = z;

}

return null;

}

}

public class ShaderProgram {

private final int programId;

public ShaderProgram(String vertexPath, String fragmentPath) {

programId = glCreateProgram();

int vertexShader = createShader(vertexPath, GL\_VERTEX\_SHADER);

int fragmentShader = createShader(fragmentPath, GL\_FRAGMENT\_SHADER);

glAttachShader(programId, vertexShader);

glAttachShader(programId, fragmentShader);

glLinkProgram(programId);

if (glGetProgrami(programId, GL\_LINK\_STATUS) == GL\_FALSE) {

throw new RuntimeException("Error al vincular programa: " + glGetProgramInfoLog(programId));

}

glDeleteShader(vertexShader);

glDeleteShader(fragmentShader);

}

private int createShader(String path, int type) {

String code = "";

try {

code = new String(Files.readAllBytes(Paths.get(path)));

} catch (IOException e) {

e.printStackTrace();

}

int shaderId = glCreateShader(type);

glShaderSource(shaderId, code);

glCompileShader(shaderId);

if (glGetShaderi(shaderId, GL\_COMPILE\_STATUS) == GL\_FALSE) {

throw new RuntimeException("Error en shader " + path + ": " + glGetShaderInfoLog(shaderId));

}

return shaderId;

}

public void use() {

glUseProgram(programId);

}

public void setUniformMat4(String name, Matrix4f matrix) {

int location = glGetUniformLocation(programId, name);

try (MemoryStack stack = MemoryStack.stackPush()) {

FloatBuffer fb = stack.mallocFloat(16);

matrix.get(fb);

glUniformMatrix4fv(location, false, fb);

}

}

public void setUniform3f(String name, Vector3f value) {

int location = glGetUniformLocation(programId, name);

if (location != -1) {

glUniform3f(location, value.x, value.y, value.z);

} else {

System.err.println("Uniform not found: " + name);

}

}

public void setUniform1i(String name, int value) {

int location = glGetUniformLocation(programId, name);

if (location == -1) {

System.err.println("Warning: uniform '" + name + "' doesn't exist!");

return;

}

glUniform1i(location, value);

}

public void bind() {

glUseProgram(programId);

}

public void unbind() {

glUseProgram(0);

}

public void cleanup() {

glDeleteProgram(programId);

}

}

public class TextRenderer {

private int textureId;

private int width;

private int height;

private Font font;

private Color color;

private String currentText;

public TextRenderer(String text, Font font, Color color) {

this.font = font;

this.color = color;

updateText(text);

}

public void updateText(String newText) {

if (newText.equals(currentText)) {

return; // Evita regenerar si no cambia

}

if (textureId != 0) {

GL11.glDeleteTextures(textureId);

}

BufferedImage img = createTextImage(newText, font, color);

this.width = img.getWidth();

this.height = img.getHeight();

this.textureId = loadTextureFromBufferedImage(img);

this.currentText = newText;

}

private BufferedImage createTextImage(String text, Font font, Color color) {

BufferedImage img = new BufferedImage(256, 64, BufferedImage.TYPE\_INT\_ARGB);

Graphics2D g = img.createGraphics();

g.setComposite(AlphaComposite.Src);

g.setRenderingHint(RenderingHints.KEY\_TEXT\_ANTIALIASING, RenderingHints.VALUE\_TEXT\_ANTIALIAS\_ON);

g.setFont(font);

g.setColor(new Color(255, 255, 255, 255)); // blanco opaco

g.setBackground(new Color(0, 0, 0, 0)); // fondo completamente transparente

g.clearRect(0, 0, img.getWidth(), img.getHeight());

g.drawString(text, 10, 40);

g.dispose();

return img;

}

public static int loadTextureFromBufferedImage(BufferedImage image) {

int[] pixels = new int[image.getWidth() \* image.getHeight()];

image.getRGB(0, 0, image.getWidth(), image.getHeight(), pixels, 0, image.getWidth());

ByteBuffer buffer = BufferUtils.createByteBuffer(image.getWidth() \* image.getHeight() \* 4);

for (int y = 0; y < image.getHeight(); y++) {

for (int x = 0; x < image.getWidth(); x++) {

int pixel = pixels[y \* image.getWidth() + x];

buffer.put((byte) ((pixel >> 16) & 0xFF)); // Red

buffer.put((byte) ((pixel >> 8) & 0xFF)); // Green

buffer.put((byte) (pixel & 0xFF)); // Blue

buffer.put((byte) ((pixel >> 24) & 0xFF)); // Alpha

}

}

buffer.flip();

int textureID = glGenTextures();

glBindTexture(GL\_TEXTURE\_2D, textureID);

// Preservar el canal alfa

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA8, image.getWidth(), image.getHeight(), 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, buffer);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glBindTexture(GL\_TEXTURE\_2D, 0);

return textureID;

}

public void render(float x, float y) {

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, textureId);

glColor4f(1, 1, 1, 1);

glBegin(GL\_QUADS);

glTexCoord2f(0, 0);

glVertex2f(x, y);

glTexCoord2f(1, 0);

glVertex2f(x + width, y);

glTexCoord2f(1, 1);

glVertex2f(x + width, y + height);

glTexCoord2f(0, 1);

glVertex2f(x, y + height);

glEnd();

glBindTexture(GL\_TEXTURE\_2D, 0);

glDisable(GL\_TEXTURE\_2D);

}

public void cleanup() {

if (textureId != 0) {

glDeleteTextures(textureId);

}

}

}

public class Texture {

private final int id;

public Texture(String fileName) throws Exception {

id = glGenTextures();

glBindTexture(GL\_TEXTURE\_2D, id);

// Parámetros de textura (filtrado y wrap)

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

// Cargar imagen

try (MemoryStack stack = MemoryStack.stackPush()) {

IntBuffer width = stack.mallocInt(1);

IntBuffer height = stack.mallocInt(1);

IntBuffer channels = stack.mallocInt(1);

// La ruta debe ser relativa al directorio del proyecto o ruta absoluta

ByteBuffer image = STBImage.stbi\_load(fileName, width, height, channels, 4);

if (image == null) {

throw new Exception("Failed to load a texture file!" + System.lineSeparator() + STBImage.stbi\_failure\_reason());

}

int texWidth = width.get(0);

int texHeight = height.get(0);

System.out.println("Texture loaded: " + texWidth + "x" + texHeight);

// Subir textura a OpenGL

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, width.get(0), height.get(0), 0,

GL\_RGBA, GL\_UNSIGNED\_BYTE, image);

STBImage.stbi\_image\_free(image);

}

}

public int getId() {

return id;

}

public void bind() {

glBindTexture(GL\_TEXTURE\_2D, id);

}

public void cleanup() {

glDeleteTextures(id);

}

}

public class Window {

private final int width;

private final int height;

private final String title;

private long window;

public Window(int width, int height, String title) {

this.width = width;

this.height = height;

this.title = title;

}

public void init() {

// Setup error callback

GLFWErrorCallback.createPrint(System.err).set();

// Init GLFW

if (!glfwInit()) {

throw new IllegalStateException("Unable to initialize GLFW");

}

// Configure GLFW

glfwDefaultWindowHints();

glfwWindowHint(GLFW\_VISIBLE, GLFW\_FALSE); // keep window hidden until after creation

glfwWindowHint(GLFW\_RESIZABLE, GLFW\_TRUE);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GLFW\_TRUE); // for macOS

// Create the window

window = glfwCreateWindow(width, height, title, NULL, NULL);

if (window == NULL) {

throw new RuntimeException("Failed to create the GLFW window");

}

// Center the window

GLFWVidMode vidmode = glfwGetVideoMode(glfwGetPrimaryMonitor());

if (vidmode != null) {

glfwSetWindowPos(

window,

(vidmode.width() - width) / 2,

(vidmode.height() - height) / 2

);

}

// Make the OpenGL context current

glfwMakeContextCurrent(window);

// Enable v-sync

glfwSwapInterval(1);

// Show the window

glfwShowWindow(window);

// Create OpenGL capabilities (must be done after context is current)

GL.createCapabilities();

// Configure OpenGL defaults

glEnable(GL\_DEPTH\_TEST); // important for 3D

glClearColor(0.1f, 0.2f, 0.3f, 1.0f); // nice blue background

glViewport(0, 0, width, height); // set viewport size

}

public void update() {

glfwSwapBuffers(window); // swap the color buffers

glfwPollEvents(); // poll for window events

}

public boolean shouldClose() {

return glfwWindowShouldClose(window);

}

public void cleanup() {

glfwDestroyWindow(window);

glfwTerminate();

glfwSetErrorCallback(null).free();

}

}

//Fragment.glsl

#version 330 core

in vec2 fragTexCoord;

in vec3 fragPos;

in vec3 normal;

in float shadeFactor;

in float vAO;

out vec4 fragColor;

uniform sampler2D textureSampler;

// Dirección de la luz (normalizada), ej: vec3(-0.5, -1.0, -0.5)

uniform vec3 lightPos;

void main() {

vec3 norm = normalize(normal);

// Dirección opuesta a la luz (porque viene "desde el sol")

float diff = max(dot(norm, -lightPos), 0.0);

diff\*=shadeFactor;

vec3 ambient = vec3(0.3)\* diff; // Luz ambiental constante

vec3 diffuse = vec3(0.7);

vec3 texColor = texture(textureSampler, fragTexCoord).rgb;

vec3 shadeDiffuse = diffuse\*shadeFactor;

vec3 lighting = (ambient + shadeDiffuse)\*texColor\*vAO;

fragColor = vec4(lighting, 1.0);

}

//selector\_fragment.glsl

#version 330 core

out vec4 fragColor;

void main() {

fragColor = vec4(1.0, 1.0, 1.0, 1.0); // Blanco puro

}

//selector\_vertex.glsl

#version 330 core

layout(location = 0) in vec3 inPosition;

uniform mat4 mvp;

void main() {

gl\_Position = mvp \* vec4(inPosition, 1.0);

}

//text\_fragment.glsl

#version 330 core

in vec2 TexCoords;

out vec4 FragColor;

uniform sampler2D text; // atlas de caracteres (un solo canal R)

uniform vec3 textColor; // color del texto (RGB)

void main() {

// La textura 'text' tiene sólo el canal R con la información de la glyph

float alpha = texture(text, TexCoords).r;

// Usamos textColor para RGB, y alpha para transparencia

FragColor = vec4(textColor, alpha);

}

//text\_vertex.glsl

#version 330 core

layout (location = 0) in vec4 vertex; // <vec2 pos, vec2 tex>

out vec2 TexCoords;

uniform mat4 projection;

void main() {

// vertex.xy = posición en pantalla (ya en coordenadas ortográficas)

// vertex.zw = coordenadas de textura

gl\_Position = projection \* vec4(vertex.xy, 0.0, 1.0);

TexCoords = vertex.zw;

}

//shader.glsl

#version 330 core

layout(location = 0) in vec3 inPosition;

layout(location = 1) in vec3 inNormal;

layout(location = 2) in vec2 inTexCoord;

layout(location = 3) in float inShade;

layout(location = 4) in float inAO;

uniform mat4 model;

uniform mat4 view;

uniform mat4 projection;

out vec2 fragTexCoord;

out vec3 fragPos;

out vec3 normal;

out float shadeFactor;

out float vAO;

void main() {

fragTexCoord = inTexCoord;

vec4 worldPos = model \* vec4(inPosition, 1.0);

fragPos = worldPos.xyz;

normal = mat3(transpose(inverse(model))) \* inNormal;

shadeFactor = inShade;

vAO = inAO;

gl\_Position = projection \* view \* worldPos;

}