#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

#define SCREEN\_WIDTH 128

#define SCREEN\_HEIGHT 64

#define OLED\_RESET    -1

#define SCREEN\_ADDRESS 0x3C

//#define BUZZER\_PIN -1  // 如果有蜂鸣器，设置为引脚号，否则设为-1

#define BUZZER\_PIN 5  // 改成你连接的GPIO引脚

Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, OLED\_RESET);

// 您的位图数据

const unsigned char bitmap[] PROGMEM = {

  0x3C, 0x00, 0x03, 0xF0, 0x00, 0x04, 0x08, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

  0xE0, 0x00, 0x0E, 0x1E, 0x00, 0x08, 0x08, 0x10, 0x00, 0x00, 0x04, 0x00, 0x04, 0x00, 0x00, 0xC0,

  0x80, 0x00, 0xF8, 0x03, 0xC0, 0x08, 0x08, 0x10, 0x00, 0x00, 0x04, 0x00, 0x0C, 0x00, 0x00, 0xC0,

  0x00, 0x03, 0x80, 0x30, 0x70, 0x08, 0x08, 0x10, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x00, 0xC0,

  0x00, 0x1E, 0x00, 0x60, 0x11, 0x08, 0x18, 0x10, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x03, 0xF0,

  0x00, 0xF0, 0x00, 0xC0, 0x10, 0x08, 0x60, 0x10, 0x00, 0x70, 0x00, 0x00, 0x00, 0x00, 0x07, 0xF0,

  0x0F, 0x80, 0x00, 0x80, 0x10, 0x09, 0x80, 0x10, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0,

  0x38, 0x00, 0xF8, 0x80, 0x10, 0x0E, 0x00, 0x70, 0x00, 0x00, 0x00, 0x07, 0xFF, 0x00, 0x00, 0xC0,

  0x60, 0x01, 0x88, 0x7E, 0x10, 0x08, 0x01, 0xC0, 0x00, 0x00, 0x00, 0x0F, 0xFF, 0x80, 0x00, 0x80,

  0x00, 0x03, 0x08, 0x03, 0x10, 0x00, 0x0F, 0x40, 0x00, 0x00, 0x00, 0x1C, 0x01, 0xC0, 0x10, 0x00,

  0x00, 0x42, 0x08, 0x01, 0x10, 0x00, 0xF8, 0x40, 0x00, 0x00, 0x00, 0x38, 0x00, 0xE0, 0x10, 0x00,

  0x00, 0xC2, 0x18, 0x23, 0x10, 0x03, 0x88, 0x40, 0x00, 0x00, 0x00, 0x70, 0x00, 0x70, 0x10, 0x00,

  0x00, 0x82, 0x71, 0x1E, 0x10, 0x1F, 0x08, 0x40, 0x00, 0x00, 0x00, 0xE0, 0x00, 0x38, 0x30, 0x00,

  0x00, 0x83, 0xC3, 0x00, 0x10, 0x71, 0x09, 0xC0, 0x02, 0x00, 0x01, 0xC0, 0x00, 0x00, 0x00, 0x00,

  0x21, 0x81, 0x86, 0x00, 0x13, 0xC1, 0x0F, 0xC0, 0x07, 0x00, 0x03, 0x80, 0x00, 0x00, 0x00, 0x00,

  0x31, 0x00, 0xFC, 0x00, 0x1E, 0x41, 0xFE, 0x40, 0x02, 0x00, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00,

  0x01, 0x00, 0x00, 0x00, 0x78, 0x43, 0xF8, 0x40, 0x00, 0x00, 0x03, 0x18, 0x00, 0x00, 0x00, 0x00,

  0x09, 0x81, 0x00, 0x00, 0xF0, 0x47, 0x08, 0x40, 0x00, 0x00, 0x03, 0x0C, 0x00, 0x00, 0x0E, 0x00,

  0x08, 0x83, 0x00, 0x03, 0x90, 0x7F, 0x08, 0x40, 0x00, 0x00, 0x03, 0x00, 0x01, 0x00, 0x11, 0x00,

  0x0C, 0xC6, 0x00, 0x1F, 0x10, 0xF1, 0x08, 0x40, 0x00, 0x00, 0x03, 0x00, 0x03, 0x80, 0x24, 0x80,

  0x06, 0x7C, 0x00, 0x7A, 0x13, 0xC1, 0x08, 0x40, 0x00, 0x00, 0x43, 0x00, 0x01, 0x00, 0x2C, 0x80,

  0x02, 0x00, 0x1F, 0xC6, 0x1E, 0x41, 0x08, 0x40, 0x00, 0x00, 0x63, 0x00, 0x00, 0x00, 0x28, 0x80,

  0x03, 0x00, 0x3C, 0x8A, 0x70, 0x41, 0x09, 0xE0, 0x10, 0x00, 0x23, 0x00, 0x00, 0x00, 0x20, 0x80,

  0x00, 0x01, 0xE1, 0x57, 0xD0, 0x41, 0x0F, 0x80, 0x10, 0x00, 0x03, 0x00, 0x00, 0x00, 0x20, 0x80,

  0x00, 0x0F, 0x22, 0x25, 0x10, 0x41, 0x3C, 0x00, 0x28, 0x00, 0x03, 0x1F, 0xFF, 0x00, 0x11, 0x00,

  0x00, 0xF8, 0x54, 0x3E, 0x10, 0x43, 0xE0, 0x00, 0x00, 0x00, 0x03, 0x3F, 0xFF, 0x83, 0x0A, 0x00,

  0x07, 0x88, 0x88, 0xE4, 0x10, 0x4F, 0x00, 0x00, 0x00, 0x00, 0x03, 0x70, 0x01, 0xC3, 0x04, 0x00,

  0x3C, 0x15, 0x07, 0x04, 0x10, 0x5B, 0x00, 0x80, 0x00, 0x00, 0x03, 0xE0, 0x00, 0xE0, 0x0E, 0x00,

  0xFA, 0x22, 0x1E, 0x04, 0x10, 0xF1, 0x80, 0x80, 0x00, 0x08, 0x03, 0xC0, 0x00, 0x70, 0x04, 0x08,

  0x89, 0x40, 0xF2, 0x04, 0x17, 0xB0, 0xC1, 0x40, 0x00, 0x0C, 0x03, 0x86, 0x0C, 0x38, 0x04, 0x18,

  0x08, 0x8F, 0x83, 0x04, 0x3C, 0x18, 0x60, 0x00, 0x00, 0x08, 0x03, 0x0F, 0x1E, 0x18, 0x04, 0x10,

  0x08, 0x78, 0x83, 0x04, 0xE6, 0x0C, 0x30, 0x00, 0x00, 0x00, 0x03, 0x19, 0xB3, 0x18, 0x06, 0x00,

  0x0F, 0xE0, 0xC2, 0x07, 0x83, 0x06, 0x18, 0x00, 0x00, 0x00, 0x03, 0x18, 0xE3, 0x18, 0x02, 0x00,

  0x0C, 0x21, 0xC3, 0x7C, 0x01, 0x83, 0x0C, 0x00, 0x00, 0x00, 0x03, 0x18, 0x43, 0x18, 0x02, 0x00,

  0x18, 0x21, 0x87, 0xC0, 0x00, 0xC1, 0x86, 0x00, 0x00, 0x00, 0x03, 0x18, 0x0B, 0x18, 0x01, 0x00,

  0x10, 0x60, 0x8F, 0x80, 0x10, 0x60, 0xC3, 0x00, 0x01, 0x00, 0x03, 0x18, 0x1B, 0x18, 0x01, 0x00,

  0x10, 0x60, 0xFB, 0x80, 0x10, 0x30, 0x61, 0x80, 0x01, 0x00, 0x03, 0x18, 0x13, 0x18, 0x01, 0x00,

  0x50, 0x21, 0xC1, 0xC0, 0x28, 0x18, 0x30, 0xC0, 0x02, 0x80, 0x03, 0x18, 0x03, 0x18, 0x01, 0x00,

  0x70, 0x3E, 0x01, 0xC0, 0x00, 0x0C, 0x18, 0x71, 0xC4, 0x40, 0x03, 0x0C, 0x06, 0x18, 0x80, 0x00,

  0x70, 0xF0, 0x00, 0xE0, 0x00, 0x06, 0x0C, 0x3B, 0x98, 0x30, 0x03, 0x06, 0x0C, 0x19, 0x81, 0x00,

  0x77, 0x38, 0x00, 0xF0, 0x00, 0x03, 0x06, 0x1E, 0x04, 0x40, 0x23, 0x03, 0x18, 0x18, 0x80, 0x00,

  0x3C, 0x38, 0x03, 0xC0, 0x00, 0x41, 0x83, 0x78, 0x02, 0x80, 0x23, 0x81, 0xB0, 0x38, 0x01, 0x00,

  0xF0, 0x1C, 0x0E, 0x00, 0x40, 0x40, 0xC1, 0xF0, 0x01, 0x00, 0x21, 0xC0, 0xE0, 0x70, 0x14, 0x60,

  0x80, 0x0C, 0x7C, 0x00, 0x40, 0xA0, 0x67, 0xC0, 0x01, 0x00, 0x60, 0xE0, 0x40, 0xE0, 0x01, 0x00,

  0x00, 0x0F, 0xE0, 0x00, 0xA0, 0x00, 0x3F, 0x00, 0x00, 0x00, 0x00, 0x70, 0x01, 0xC0, 0x00, 0x00,

  0x00, 0x07, 0x80, 0x00, 0x00, 0x00, 0xFC, 0x00, 0x00, 0x00, 0x00, 0x3F, 0xFF, 0x88, 0x02, 0x80,

  0x00, 0x06, 0x00, 0x00, 0x00, 0x01, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x1F, 0xFF, 0x0C, 0x08, 0x40,

  0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,

  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

  0x00, 0x00, 0x00, 0x03, 0x80, 0xFE, 0x10, 0x18, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x80,

  0x0C, 0x0F, 0x1F, 0x87, 0xE0, 0xFE, 0x18, 0x31, 0x00, 0xF0, 0x3F, 0x8F, 0xE2, 0x20, 0x03, 0x04,

  0x0C, 0x1D, 0x1F, 0x8C, 0x60, 0xC6, 0x0C, 0x23, 0x81, 0x08, 0x24, 0x91, 0x02, 0x20, 0x1C, 0x04,

  0x00, 0x30, 0x18, 0x08, 0x30, 0xC0, 0x04, 0x65, 0x42, 0x00, 0x2E, 0xA1, 0x02, 0xFC, 0x20, 0x04,

  0x0C, 0x30, 0x18, 0x00, 0x30, 0xC0, 0x06, 0xCD, 0x62, 0x00, 0x24, 0x8F, 0xC3, 0x24, 0x22, 0x04,

  0x0C, 0x60, 0x18, 0x00, 0x30, 0xFC, 0x03, 0x95, 0x52, 0x00, 0x2E, 0x89, 0x02, 0xA8, 0x22, 0x04,

  0x0C, 0x60, 0x1F, 0x03, 0xF0, 0x7E, 0x01, 0x9F, 0xF2, 0xF0, 0x20, 0x89, 0x02, 0x30, 0x22, 0x04,

  0x0C, 0x60, 0x1F, 0x06, 0x60, 0x06, 0x03, 0x08, 0x23, 0x08, 0x24, 0xBF, 0xE3, 0xFF, 0x1F, 0xC4,

  0x0C, 0x70, 0x18, 0x0C, 0x30, 0x06, 0x06, 0x0F, 0x22, 0x08, 0x2A, 0x81, 0x0A, 0x20, 0x02, 0x00,

  0x0C, 0x30, 0x18, 0x0C, 0x30, 0xC6, 0x0C, 0x09, 0x22, 0x08, 0x4A, 0x81, 0x12, 0x20, 0x0A, 0x84,

  0x0C, 0x1D, 0x1F, 0x8E, 0x70, 0xFE, 0x18, 0x0D, 0x22, 0x08, 0x44, 0x81, 0x02, 0x50, 0x12, 0x40,

  0x0C, 0x0F, 0x1F, 0x87, 0xEC, 0xFE, 0x30, 0x09, 0x21, 0x10, 0x80, 0x81, 0x02, 0x48, 0x22, 0x30,

  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x60, 0x1F, 0xF0, 0xE1, 0x01, 0x81, 0x02, 0x87, 0x06, 0x00,

  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

};

// 效果名称

const char\* effectNames[] = {

  "TIME TUNNEL",    // 时空隧道

  "DNA HELIX",      // DNA螺旋

  "STAR FIELD",     // 星域冲击

  "PORTAL",         // 传送门

  "DECODE",         // 矩阵解码

  "SHATTER",        // 碎片重组

  "WARP SPEED",     // 曲速引擎

  "QUANTUM"         // 量子叠加

};

// 粒子系统

struct Particle {

  int x, y;

  int vx, vy;

  int targetX, targetY;

  uint8\_t life;

};

#define MAX\_PARTICLES 35

Particle particles[MAX\_PARTICLES];

// 星星

struct Star {

  int x, y, z;

};

#define MAX\_STARS 30

Star stars[MAX\_STARS];

unsigned long effectStartTime = 0;

int currentEffect = 0;

int transitionAlpha = 0;

void setup() {

  Serial.begin(115200);

  Wire.begin(6, 7);

  if(!display.begin(SSD1306\_SWITCHCAPVCC, SCREEN\_ADDRESS)) {

    Serial.println(F("初始化失败"));

    for(;;);

  }

  if (BUZZER\_PIN >= 0) {

    pinMode(BUZZER\_PIN, OUTPUT);

  }

  // 史诗级启动动画

  epicBootAnimation();

  effectStartTime = millis();

  initStars();

  Serial.println(F("╔═══════════════════════════════╗"));

  Serial.println(F("║  宇宙第一 终极显示系统 V1.0  ║"));

  Serial.println(F("║  ULTIMATE DISPLAY SYSTEM     ║"));

  Serial.println(F("╚═══════════════════════════════╝"));

}

void loop() {

  unsigned long elapsed = millis() - effectStartTime;

  // 每6秒切换效果

  if (elapsed >= 6000) {

    playTransition();  // 过渡动画

    currentEffect = (currentEffect + 1) % 8;

    effectStartTime = millis();

    elapsed = 0;

    // 重新初始化

    if (currentEffect == 2) initStars();

    if (currentEffect == 5) initShatter();

    // 打印效果名

    Serial.println();

    Serial.print(F(">>> 效果 #"));

    Serial.print(currentEffect + 1);

    Serial.print(F(": "));

    Serial.println(effectNames[currentEffect]);

    beep(100);  // 音效提示

  }

  // 执行当前效果

  switch(currentEffect) {

    case 0: effect1\_TimeTunnel(elapsed); break;

    case 1: effect2\_DNAHelix(elapsed); break;

    case 2: effect3\_StarField(elapsed); break;

    case 3: effect4\_Portal(elapsed); break;

    case 4: effect5\_MatrixDecode(elapsed); break;

    case 5: effect6\_Shatter(elapsed); break;

    case 6: effect7\_WarpSpeed(elapsed); break;

    case 7: effect8\_Quantum(elapsed); break;

  }

}

// ============ 史诗级启动动画 ============

void epicBootAnimation() {

  // 第一阶段：脉冲

  for (int i = 0; i < 3; i++) {

    display.clearDisplay();

    display.fillCircle(64, 32, 5 + i \* 8, SSD1306\_WHITE);

    display.display();

    beep(50);

    delay(100);

  }

  // 第二阶段：标题

  display.clearDisplay();

  display.setTextSize(2);

  display.setTextColor(SSD1306\_WHITE);

  display.setCursor(15, 10);

  display.println(F("ULTIMATE"));

  display.setCursor(25, 30);

  display.println(F("SYSTEM"));

  display.display();

  beep(200);

  delay(1000);

  // 第三阶段：扫描线

  for (int y = 0; y < 64; y += 2) {

    display.drawFastHLine(0, y, 128, SSD1306\_WHITE);

    display.display();

    if (y % 10 == 0) beep(10);

  }

  delay(500);

}

// ============ 过渡动画 ============

void playTransition() {

  for (int i = 0; i < 10; i++) {

    display.clearDisplay();

    // 四角向中心收缩

    int s = i \* 13;

    display.drawLine(0, 0, s, s, SSD1306\_WHITE);

    display.drawLine(127, 0, 127-s, s, SSD1306\_WHITE);

    display.drawLine(0, 63, s, 63-s, SSD1306\_WHITE);

    display.drawLine(127, 63, 127-s, 63-s, SSD1306\_WHITE);

    display.display();

    delay(30);

  }

}

// ============ 效果1: 时空隧道 ============

void effect1\_TimeTunnel(unsigned long t) {

  display.clearDisplay();

  int depth = (t / 30) % 40;

  // 绘制隧道环

  for (int i = 0; i < 8; i++) {

    int d = (depth + i \* 5) % 40;

    int r = 3 + d;

    if (r < 50) {

      display.drawCircle(64, 32, r, SSD1306\_WHITE);

      // 每隔一个画粗线

      if (i % 2 == 0) {

        display.drawCircle(64, 32, r-1, SSD1306\_WHITE);

      }

    }

  }

  // 中心图像逐渐显现

  float progress = min(1.0, t / 4000.0);

  int maxY = 64 \* progress;

  for (int y = 0; y < maxY; y++) {

    for (int x = 0; x < 128; x++) {

      int byteIdx = y \* 16 + x / 8;

      int bitIdx = 7 - (x % 8);

      if (pgm\_read\_byte(&bitmap[byteIdx]) & (1 << bitIdx)) {

        display.drawPixel(x, y, SSD1306\_WHITE);

      }

    }

  }

  display.display();

  delay(25);

}

// ============ 效果2: DNA双螺旋 ============

void effect2\_DNAHelix(unsigned long t) {

  display.clearDisplay();

  float angle = t / 500.0;

  // 绘制双螺旋

  for (int x = 0; x < 128; x += 3) {

    float a = angle + x \* 0.1;

    // 螺旋1

    int y1 = 32 + sin(a) \* 15;

    display.fillCircle(x, y1, 2, SSD1306\_WHITE);

    // 螺旋2（反相）

    int y2 = 32 - sin(a) \* 15;

    display.fillCircle(x, y2, 2, SSD1306\_WHITE);

    // 连接线

    if (abs(y1 - y2) < 5) {

      display.drawLine(x, y1, x, y2, SSD1306\_WHITE);

    }

  }

  // 叠加图像（半透明效果）

  if (t > 2000) {

    for (int y = 0; y < 64; y += 2) {

      for (int x = 0; x < 128; x += 2) {

        int byteIdx = y \* 16 + x / 8;

        int bitIdx = 7 - (x % 8);

        if (pgm\_read\_byte(&bitmap[byteIdx]) & (1 << bitIdx)) {

          display.drawPixel(x, y, SSD1306\_WHITE);

        }

      }

    }

  }

  display.display();

  delay(30);

}

// ============ 效果3: 星域冲击 ============

void initStars() {

  for (int i = 0; i < MAX\_STARS; i++) {

    stars[i].x = random(-64, 64);

    stars[i].y = random(-32, 32);

    stars[i].z = random(1, 30);

  }

}

void effect3\_StarField(unsigned long t) {

  display.clearDisplay();

  // 更新星星

  for (int i = 0; i < MAX\_STARS; i++) {

    stars[i].z -= 1;

    if (stars[i].z <= 0) {

      stars[i].x = random(-64, 64);

      stars[i].y = random(-32, 32);

      stars[i].z = 30;

    }

    // 透视投影

    int sx = 64 + (stars[i].x \* 30) / stars[i].z;

    int sy = 32 + (stars[i].y \* 30) / stars[i].z;

    if (sx >= 0 && sx < 128 && sy >= 0 && sy < 64) {

      int size = (30 - stars[i].z) / 15 + 1;

      // 绘制星星拖尾

      int oldSx = 64 + (stars[i].x \* 30) / (stars[i].z + 1);

      int oldSy = 32 + (stars[i].y \* 30) / (stars[i].z + 1);

      display.drawLine(sx, sy, oldSx, oldSy, SSD1306\_WHITE);

      display.fillCircle(sx, sy, size, SSD1306\_WHITE);

    }

  }

  // 图像从远处飞来

  if (t > 3000) {

    float scale = min(1.0, (t - 3000) / 2000.0);

    int w = 128 \* scale;

    int h = 64 \* scale;

    int ox = (128 - w) / 2;

    int oy = (64 - h) / 2;

    for (int y = 0; y < h; y += 2) {

      for (int x = 0; x < w; x += 2) {

        int sx = x / scale;

        int sy = y / scale;

        int byteIdx = sy \* 16 + sx / 8;

        int bitIdx = 7 - (sx % 8);

        if (byteIdx < 1024 && (pgm\_read\_byte(&bitmap[byteIdx]) & (1 << bitIdx))) {

          display.drawPixel(ox + x, oy + y, SSD1306\_WHITE);

        }

      }

    }

  }

  display.display();

  delay(25);

}

// ============ 效果4: 传送门 ============

void effect4\_Portal(unsigned long t) {

  display.clearDisplay();

  // 传送门圆环扩大

  int maxRadius = min(50, (int)(t / 50));

  for (int r = maxRadius; r > 0; r -= 5) {

    if ((r / 5) % 2 == 0) {

      display.drawCircle(64, 32, r, SSD1306\_WHITE);

    }

  }

  // 旋转效果

  if (t > 1000) {

    float angle = t / 1000.0;

    for (int y = 0; y < 64; y++) {

      for (int x = 0; x < 128; x++) {

        float dx = x - 64;

        float dy = y - 32;

        float dist = sqrt(dx\*dx + dy\*dy);

        if (dist < maxRadius - 10) {

          float a = atan2(dy, dx) + angle \* (1.0 - dist / maxRadius);

          int sx = 64 + cos(a) \* dist;

          int sy = 32 + sin(a) \* dist;

          if (sx >= 0 && sx < 128 && sy >= 0 && sy < 64) {

            int byteIdx = sy \* 16 + sx / 8;

            int bitIdx = 7 - (sx % 8);

            if (pgm\_read\_byte(&bitmap[byteIdx]) & (1 << bitIdx)) {

              display.drawPixel(x, y, SSD1306\_WHITE);

            }

          }

        }

      }

    }

  }

  display.display();

  delay(30);

}

// ============ 效果5: 矩阵解码 ============

void effect5\_MatrixDecode(unsigned long t) {

  display.clearDisplay();

  // 从左到右解码

  int decodeX = (t / 30) % 140;

  for (int y = 0; y < 64; y++) {

    for (int x = 0; x < 128; x++) {

      int byteIdx = y \* 16 + x / 8;

      int bitIdx = 7 - (x % 8);

      bool pixel = pgm\_read\_byte(&bitmap[byteIdx]) & (1 << bitIdx);

      if (x < decodeX) {

        // 已解码区域

        if (pixel) display.drawPixel(x, y, SSD1306\_WHITE);

      } else if (x == decodeX) {

        // 扫描线

        display.drawPixel(x, y, SSD1306\_WHITE);

      } else {

        // 未解码区域 - 随机噪点

        if (random(10) < 3 && pixel) {

          display.drawPixel(x, y, SSD1306\_WHITE);

        }

      }

    }

  }

  // 扫描线发光

  for (int i = -3; i <= 3; i++) {

    int x = decodeX + i;

    if (x >= 0 && x < 128) {

      display.drawFastVLine(x, 0, 64, SSD1306\_WHITE);

    }

  }

  display.display();

  delay(20);

}

// ============ 效果6: 碎片重组 ============

struct Fragment {

  int x, y, tx, ty, vx, vy;

};

#define MAX\_FRAGS 20

Fragment frags[MAX\_FRAGS];

void initShatter() {

  for (int i = 0; i < MAX\_FRAGS; i++) {

    frags[i].tx = (i % 5) \* 25;

    frags[i].ty = (i / 5) \* 16;

    frags[i].x = random(-50, 178);

    frags[i].y = random(-30, 94);

    frags[i].vx = 0;

    frags[i].vy = 0;

  }

}

void effect6\_Shatter(unsigned long t) {

  display.clearDisplay();

  bool gathering = t > 2000;

  for (int i = 0; i < MAX\_FRAGS; i++) {

    if (gathering) {

      frags[i].vx = (frags[i].tx - frags[i].x) / 8;

      frags[i].vy = (frags[i].ty - frags[i].y) / 8;

    }

    frags[i].x += frags[i].vx;

    frags[i].y += frags[i].vy;

    // 绘制碎片

    int fx = frags[i].x;

    int fy = frags[i].y;

    for (int dy = 0; dy < 16 && fy + dy < 64; dy++) {

      for (int dx = 0; dx < 25 && fx + dx < 128; dx++) {

        int sy = frags[i].ty + dy;

        int sx = frags[i].tx + dx;

        if (sy >= 0 && sy < 64 && sx >= 0 && sx < 128) {

          int byteIdx = sy \* 16 + sx / 8;

          int bitIdx = 7 - (sx % 8);

          if (pgm\_read\_byte(&bitmap[byteIdx]) & (1 << bitIdx)) {

            int py = fy + dy;

            int px = fx + dx;

            if (px >= 0 && px < 128 && py >= 0 && py < 64) {

              display.drawPixel(px, py, SSD1306\_WHITE);

            }

          }

        }

      }

    }

    // 绘制碎片边框

    display.drawRect(fx, fy, 25, 16, SSD1306\_WHITE);

  }

  display.display();

  delay(25);

}

// ============ 效果7: 曲速引擎 ============

void effect7\_WarpSpeed(unsigned long t) {

  display.clearDisplay();

  // 高速线条

  for (int i = 0; i < 40; i++) {

    int x = (t \* 5 + i \* 20) % 160 - 32;

    int y = random(0, 64);

    int len = random(10, 40);

    if (x >= 0 && x < 128) {

      display.drawFastHLine(x, y, min(len, 128-x), SSD1306\_WHITE);

    }

  }

  // 图像拉伸效果

  for (int y = 0; y < 64; y++) {

    int stretch = sin(t / 300.0 + y \* 0.1) \* 5;

    for (int x = 0; x < 128; x++) {

      int sx = (x - stretch + 128) % 128;

      int byteIdx = y \* 16 + sx / 8;

      int bitIdx = 7 - (sx % 8);

      if (pgm\_read\_byte(&bitmap[byteIdx]) & (1 << bitIdx)) {

        display.drawPixel(x, y, SSD1306\_WHITE);

      }

    }

  }

  display.display();

  delay(20);

}

// ============ 效果8: 量子叠加 ============

void effect8\_Quantum(unsigned long t) {

  display.clearDisplay();

  // 多重叠加图像

  int numLayers = 3;

  for (int layer = 0; layer < numLayers; layer++) {

    int offsetX = sin(t / 500.0 + layer \* 2) \* 10;

    int offsetY = cos(t / 600.0 + layer \* 2) \* 5;

    // 只绘制部分像素实现透明效果

    for (int y = 0; y < 64; y += (layer + 1)) {

      for (int x = 0; x < 128; x += (layer + 1)) {

        int sx = x - offsetX;

        int sy = y - offsetY;

        if (sx >= 0 && sx < 128 && sy >= 0 && sy < 64) {

          int byteIdx = sy \* 16 + sx / 8;

          int bitIdx = 7 - (sx % 8);

          if (pgm\_read\_byte(&bitmap[byteIdx]) & (1 << bitIdx)) {

            display.drawPixel(x, y, SSD1306\_WHITE);

          }

        }

      }

    }

  }

  // 量子波动（闪烁边框）

  if ((t / 100) % 3 == 0) {

    display.drawRect(0, 0, 128, 64, SSD1306\_WHITE);

    display.drawRect(2, 2, 124, 60, SSD1306\_WHITE);

  }

  // 呼吸灯

  int brightness = (sin(t / 300.0) \* 127.5) + 127.5;

  display.ssd1306\_command(SSD1306\_SETCONTRAST);

  display.ssd1306\_command(brightness);

  display.display();

  delay(35);

}

// ============ 辅助函数 ============

void beep(int duration) {

  if (BUZZER\_PIN >= 0) {

    digitalWrite(BUZZER\_PIN, HIGH);

    delay(duration);

    digitalWrite(BUZZER\_PIN, LOW);

  }

}