Arquitectura de Computadoras

Trabajo Práctico Especial

ARES - ARES Recursive Experimental System

Implementación del Juego TRON

Autores:
Juan Ignacio Raggio
Enzo Canelo
Matias Sanchez

Segundo Cuatrimestre 2025
18/10/2025

Índice

1 Resumen Ejecutivo

El proyecto ARES (ARES Recursive Experimental System) consiste en la implementación de un kernel de sistema operativo booteable basado en Pure64, diseñado para arquitectura Intel x86-64 en modo largo (Long Mode). El objetivo principal es crear un entorno interactivo que administre recursos de hardware y proporcione una API para aplicaciones de usuario, con el juego TRON como aplicación insignia.

Importante: Este proyecto está completamente desarrollado en C y Assembly x86-64, sin utilizar librerías externas. Todo el código, desde el bootloader hasta las aplicaciones de usuario, es implementado desde cero.

El sistema se estructura en dos espacios claramente separados:

- **Kernel Space**: Gestiona directamente el hardware mediante drivers (teclado, video, timer, sonido) y proporciona servicios al espacio de usuario.
- **User Space**: Contiene aplicaciones como la shell interactiva y el juego TRON, que acceden al hardware únicamente a través de syscalls.

La implementación del juego TRON servirá como demostración de las capacidades del sistema, incluyendo gráficos 2D, manejo de entrada de usuario, detección de colisiones y física básica. Además, se plantea una extensión futura a 3D como proyecto expandible.

2 Planificación e Implementación del Proyecto

2.1 Fase 1: Infraestructura Base del Kernel

2.1.1 Bootloader y Modo Protegido

El proyecto utiliza Pure64 como base del bootloader, que debe ser configurado para:

- Arrancar el sistema en modo largo (64-bit)
- Detectar la resolución de video disponible mediante EDID
- Configurar el framebuffer VESA para modo gráfico
- Cargar el kernel en memoria

Nota: El código actual en `isa.asm` y `sysvar.asm` ya implementa parcialmente la detección de resoluciones VESA. Soporta múltiples resoluciones: 1024×768, 1366×768, 1024×600 en formatos de 24bpp y 32bpp.

```
; Ejemplo de configuración VESA del código actual
resoluciones_preferidas:
    dw 1024, 768, 24, 0x0000
    dw 1024, 768, 32, 0x0000
    dw 1366, 768, 32, 0x0000
    dw 1024, 600, 32, 0x0000
```

2.1.2 Drivers Básicos

2.1.2.1 Driver de Teclado

Debe implementar:

- Manejo de IRQ1 (interrupción de teclado)
- Traducción de scan codes a caracteres ASCII
- Buffer circular para almacenar teclas presionadas
- Soporte para teclas especiales (flechas, WASD, Enter, Esc)

```
// Estructura propuesta para el driver de teclado
typedef struct {
    uint8_t buffer[256];
    uint8_t read_pos;
    uint8_t write_pos;
    uint8_t modifiers;  // Shift, Ctrl, Alt
} keyboard_state_t;

void keyboard_handler(void);
char keyboard_get_char(void);
bool keyboard_check_key(uint8_t scancode);
```

2.1.2.2 Driver de Video

El driver de video debe proporcionar primitivas gráficas básicas:

- Acceso directo al framebuffer VESA
- Funciones para pintar píxeles individuales
- Dibujo de líneas (algoritmo de Bresenham)
- Dibujo de rectángulos (rellenos y sin rellenar)
- Limpieza de pantalla
- Double buffering para evitar parpadeo

```
// API propuesta del driver de video
typedef struct {
    uint32_t* framebuffer;
    uint16_t width;
    uint16_t height;
    uint8_t bpp;
    uint32_t pitch;
} video_info_t;

void video_init(void);
void video_put_pixel(uint16_t x, uint16_t y, uint32_t color);
void video_draw_line(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint32_t
color);
void video_draw_rect(uint16_t x, uint16_t y, uint16_t w, uint16_t h, uint32_t color);
void video_clear(uint32_t color);
void video_swap_buffers(void);
```

2.1.2.3 Driver de Timer/RTC

Necesario para:

- Control de velocidad del juego
- Medición de FPS
- Timestamps para benchmarking
- Delays precisos

El código base en `isa.asm` ya configura el RTC a 1024Hz:

```
mov al, 00100110b  ; UIP (0), RTC@32.768KHz (010), Rate@1024Hz (0110)
out 0x71, al
```

2.1.2.4 Driver de Sonido (PC Speaker)

Implementación básica para:

- Beeps en eventos del juego
- Efectos de sonido simples
- Frecuencias programables

```
void speaker_play(uint16_t frequency, uint32_t duration_ms);
void speaker_stop(void);
```

2.1.3 Gestión de Memoria

2.1.3.1 Memory Map (E820)

El sistema utiliza el mapa de memoria E820 provisto por el BIOS para conocer las regiones de memoria disponibles. Ya implementado en `isa.asm`.

2.1.3.2 Allocator Básico

Implementar un allocador simple tipo bump allocator o buddy allocator:

```
void* kmalloc(size_t size);
void kfree(void* ptr);
void* krealloc(void* ptr, size_t new_size);
```

2.1.3.3 MMU (Memory Management Unit)

Configurar page tables para:

- Separación kernel space (ring 0) / user space (ring 3)
- Identity mapping para el kernel
- Mapeo del framebuffer a memoria
- Protección de memoria

Importante: La arquitectura x86-64 requiere paginación de 4 niveles: PML4, PDPT, PD, PT. Cada entrada de tabla es de 8 bytes.

2.1.4 Sistema de Interrupciones

2.1.4.1 IDT (Interrupt Descriptor Table)

Configurar 256 entradas para:

- Excepciones del CPU (0-31)
- IRQs de hardware (32-47)
- Syscalls (0x80 o usando SYSCALL instruction)

```
typedef struct {
    uint16_t offset_low;
    uint16_t selector;
    uint8_t ist;
    uint8_t type_attr;
    uint16_t offset_mid;
    uint32_t offset_high;
    uint32_t zero;
} __attribute__((packed)) idt_entry_t;

void idt_init(void);
void idt_set_gate(uint8_t num, uint64_t handler, uint8_t flags);
```

2.1.4.2 Exception Handlers

Implementar handlers para excepciones críticas:

- Division por cero (#DE)
- Invalid opcode (#UD)
- General protection fault (#GP)
- Page fault (#PF)

Cada handler debe:

1. Guardar el estado de todos los registros
2. Imprimir información de debug (RIP, RSP, error code)
3. Mostrar un stack trace si es posible
4. Retornar a la shell (no colgar el sistema)

Tip: Para debugging, implementar un volcado hexadecimal de memoria alrededor de RIP para ver el código que causó la excepción.

2.2 Fase 2: User Space y API del Kernel

2.2.1 Sistema de Syscalls

Implementar el mecanismo de syscalls usando `int 0x80` o la instrucción `SYSCALL` de x86-64:

```
; Handler de syscall en Assembly
syscall_handler:
    ; RAX = número de syscall
    ; RDI, RSI, RDX, R10, R8, R9 = argumentos (System V ABI)

    push rbp
    mov rbp, rsp

    ; Validar número de syscall
    cmp rax, SYSCALL_MAX
    jae .invalid

    ; Llamar a la función correspondiente
    call [syscall_table + rax*8]

    pop rbp
    iretq

.invalid:
    mov rax, -1  ; Error
    pop rbp
    iretq
```

2.2.2 API Básica del Kernel

Implementar las siguientes syscalls fundamentales:

2.2.2.1 I/O Básico

```
// Syscall 0: read
ssize_t sys_read(int fd, void* buf, size_t count);

// Syscall 1: write
ssize_t sys_write(int fd, const void* buf, size_t count);

// Syscall 2: getchar
int sys_getchar(void);

// Syscall 3: putchar
int sys_putchar(int c);
```

2.2.2.2 I/O Formateado

```
// Syscall 4: printf
int sys_printf(const char* format, ...);

// Syscall 5: scanf
int sys_scanf(const char* format, ...);
```

Nota: Para implementar printf/scanf sin librerías externas, se necesita un parser de format strings y conversión de números a strings (itoa, ftoa, etc).

2.2.2.3 Video

```
// Syscall 10: draw_pixel
void sys_draw_pixel(uint16_t x, uint16_t y, uint32_t color);

// Syscall 11: draw_line
void sys_draw_line(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint32_t color);

// Syscall 12: clear_screen
void sys_clear_screen(uint32_t color);

// Syscall 13: swap_buffers
void sys_swap_buffers(void);

// Syscall 14: get_screen_info
void sys_get_screen_info(uint16_t* width, uint16_t* height);
```

2.2.2.4 Input

```
// Syscall 20: get_key
int sys_get_key(void);

// Syscall 21: check_key
bool sys_check_key(uint8_t scancode);
```

2.2.2.5 Sistema

```
// Syscall 30: get_ticks
uint64_t sys_get_ticks(void);

// Syscall 31: sleep
void sys_sleep(uint32_t ms);

// Syscall 32: get_cpu_info
void sys_get_cpu_info(cpu_info_t* info);

// Syscall 33: get_mem_info
void sys_get_mem_info(mem_info_t* info);
```

2.2.3 Shell Interactiva

La shell es el programa principal que se ejecuta al arrancar el sistema.

2.2.3.1 Características

- Prompt personalizable (ej: `ARES>`)
- Parser de comandos con argumentos
- Historial de comandos (opcional)
- Auto-completado (opcional)

2.2.3.2 Comandos Básicos

```
help       - Muestra lista de comandos disponibles
time       - Muestra hora del sistema (RTC)
meminfo    - Muestra registros del procesador y memoria
cpuinfo    - Información del CPU
clear      - Limpia la pantalla
tron       - Lanza el juego TRON
benchmark  - Ejecuta benchmarks del sistema
```

2.2.3.3 Implementación del Parser

```
typedef struct {
    char* name;
    void (*function)(int argc, char** argv);
    char* description;
} command_t;

command_t commands[] = {
    {"help", cmd_help, "Muestra comandos disponibles"},
    {"time", cmd_time, "Muestra hora del sistema"},
    {"meminfo", cmd_meminfo, "Info de memoria y registros"},
    {"tron", cmd_tron, "Lanza el juego TRON"},
    // ...
};

void shell_execute(char* cmdline) {
```

```
    int argc;
    char* argv[MAX_ARGS];

    // Parsear comando y argumentos
    argc = parse_command(cmdline, argv);

    if (argc == 0) return;

    // Buscar comando
    for (int i = 0; i < NUM_COMMANDS; i++) {
        if (strcmp(argv[0], commands[i].name) == 0) {
            commands[i].function(argc, argv);
            return;
        }
    }

    printf("Comando no encontrado: %s\n", argv[0]);
}
```

2.3 Fase 3: Implementación del Juego TRON (2D)

2.3.1 Motor Gráfico 2D

2.3.1.1 Sistema de Coordenadas

El juego TRON se implementa sobre una grilla lógica que se mapea al framebuffer:

- Grid lógico: Por ejemplo 128×96 celdas
- Cada celda representa una posición del jugador
- El trail (rastro) se dibuja como rectángulos en el framebuffer

```
#define GRID_WIDTH 128
#define GRID_HEIGHT 96

typedef struct {
    uint8_t grid[GRID_HEIGHT][GRID_WIDTH];  // 0=vacío, 1=P1, 2=P2, 3=pared
} game_grid_t;

void grid_to_screen(uint8_t gx, uint8_t gy, uint16_t* sx, uint16_t* sy) {
    *sx = (gx * screen_width) / GRID_WIDTH;
    *sy = (gy * screen_height) / GRID_HEIGHT;
}
```

2.3.1.2 Double Buffering

Para evitar flickering:

```
uint32_t* front_buffer;
uint32_t* back_buffer;

void render_frame(void) {
```

```
    // Dibujar todo en back_buffer
    draw_game_to_buffer(back_buffer);

    // Swap
    sys_swap_buffers();
}
```

2.3.1.3 Paleta de Colores

```
#define COLOR_BACKGROUND  0x001a1a2e
#define COLOR_PLAYER1     0x0000ff88  // Verde/Cian
#define COLOR_PLAYER2     0x00ff00ff  // Rosa/Magenta
#define COLOR_WALL        0x00ffffff  // Blanco
#define COLOR_UI          0x00aaaaaa  // Gris
```

2.3.2 Lógica del Juego

2.3.2.1 Estructuras de Datos

```
typedef enum {
    DIR_UP,
    DIR_DOWN,
    DIR_LEFT,
    DIR_RIGHT
} direction_t;

typedef struct {
    uint8_t x;
    uint8_t y;
    direction_t direction;
    uint32_t color;
    uint8_t alive;
    uint16_t score;
} player_t;

typedef struct {
    player_t players[2];
    game_grid_t grid;
    uint8_t game_state;  // 0=menu, 1=playing, 2=gameover
    uint32_t ticks;
    uint16_t speed;      // Movimientos por segundo
} game_state_t;
```

2.3.2.2 Game Loop

```
void game_loop(void) {
    game_state_t game;
    game_init(&game);

    uint64_t last_update = sys_get_ticks();
```

```
    uint32_t ms_per_update = 1000 / game.speed;

    while (game.game_state == STATE_PLAYING) {
        uint64_t now = sys_get_ticks();

        // Input
        handle_input(&game);

        // Update
        if (now - last_update >= ms_per_update) {
            update_game(&game);
            last_update = now;
        }

        // Render
        render_game(&game);
    }

    // Game Over screen
    show_game_over(&game);
}
```

2.3.2.3 Física y Movimiento

```
void update_player(player_t* player, game_grid_t* grid) {
    // Guardar posición anterior
    uint8_t old_x = player->x;
    uint8_t old_y = player->y;

    // Mover según dirección
    switch (player->direction) {
        case DIR_UP:    player->y--; break;
        case DIR_DOWN:  player->y++; break;
        case DIR_LEFT:  player->x--; break;
        case DIR_RIGHT: player->x++; break;
    }

    // Wrap around (opcional)
    player->x %= GRID_WIDTH;
    player->y %= GRID_HEIGHT;

    // Marcar trail en la posición anterior
    grid->grid[old_y][old_x] = player_id;
}
```

2.3.2.4 Detección de Colisiones

```
bool check_collision(player_t* player, game_grid_t* grid) {
    // Verificar si la nueva posición está ocupada
    uint8_t cell = grid->grid[player->y][player->x];

    if (cell != 0) {
```

```
        // Colisionó con trail o pared
        player->alive = 0;
        return true;
    }

    return false;
}
```

2.3.3 Modos de Juego

2.3.3.1 Modo 2 Jugadores

```
void handle_input_2p(game_state_t* game) {
    // Jugador 1: WASD
    if (sys_check_key(KEY_W) && game->players[0].direction != DIR_DOWN)
        game->players[0].direction = DIR_UP;
    if (sys_check_key(KEY_S) && game->players[0].direction != DIR_UP)
        game->players[0].direction = DIR_DOWN;
    if (sys_check_key(KEY_A) && game->players[0].direction != DIR_RIGHT)
        game->players[0].direction = DIR_LEFT;
    if (sys_check_key(KEY_D) && game->players[0].direction != DIR_LEFT)
        game->players[0].direction = DIR_RIGHT;

    // Jugador 2: Flechas
    if (sys_check_key(KEY_UP) && game->players[1].direction != DIR_DOWN)
        game->players[1].direction = DIR_UP;
    if (sys_check_key(KEY_DOWN) && game->players[1].direction != DIR_UP)
        game->players[1].direction = DIR_DOWN;
    if (sys_check_key(KEY_LEFT) && game->players[1].direction != DIR_RIGHT)
        game->players[1].direction = DIR_LEFT;
    if (sys_check_key(KEY_RIGHT) && game->players[1].direction != DIR_LEFT)
        game->players[1].direction = DIR_RIGHT;
}
```

2.3.3.2 Pantalla de Selección

```
void game_menu(void) {
    sys_clear_screen(COLOR_BACKGROUND);

    // Título
    draw_text(center_x, 100, "TRON", COLOR_PLAYER1, FONT_LARGE);

    // Opciones
    draw_text(center_x, 300, "1. Un Jugador", COLOR_UI, FONT_NORMAL);
    draw_text(center_x, 350, "2. Dos Jugadores", COLOR_UI, FONT_NORMAL);
    draw_text(center_x, 400, "3. Configuración", COLOR_UI, FONT_NORMAL);

    sys_swap_buffers();

    // Esperar selección
    int choice = wait_for_key('1', '3');
```

```
    switch(choice) {
        case '1': start_single_player(); break;
        case '2': start_two_players(); break;
        case '3': show_settings(); break;
    }
}
```

2.3.4 Features Adicionales

2.3.4.1 Sistema de Velocidad

```
typedef enum {
    SPEED_SLOW = 10,    // 10 movimientos/segundo
    SPEED_NORMAL = 20,
    SPEED_FAST = 30,
    SPEED_TURBO = 60
} game_speed_t;

void apply_speed_boost(player_t* player, uint16_t* speed) {
    if (sys_check_key(KEY_SHIFT) && player->turbo_meter > 0) {
        *speed = SPEED_TURBO;
        player->turbo_meter--;
    }
}
```

2.3.4.2 Obstáculos Procedurales

```
void generate_obstacles(game_grid_t* grid, uint8_t difficulty) {
    uint32_t num_obstacles = 5 + difficulty * 3;

    for (uint32_t i = 0; i < num_obstacles; i++) {
        uint8_t x = rand() % GRID_WIDTH;
        uint8_t y = rand() % GRID_HEIGHT;
        uint8_t w = 3 + rand() % 10;
        uint8_t h = 3 + rand() % 10;

        draw_obstacle_rect(grid, x, y, w, h);
    }
}
```

2.3.4.3 Efectos de Sonido

```
void play_game_sounds(game_state_t* game) {
    // Sonido de movimiento (opcional)
    if (game->ticks % 10 == 0) {
        speaker_play(440, 10);  // A4, 10ms
    }

    // Sonido de colisión
```

```
    for (int i = 0; i < 2; i++) {
        if (!game->players[i].alive && game->players[i].just_died) {
            speaker_play(100, 200);  // Grave, 200ms
            game->players[i].just_died = 0;
        }
    }
}
```

2.3.4.4 UI y HUD

```
void render_hud(game_state_t* game) {
    char buffer[64];

    // Puntuación Jugador 1
    sprintf(buffer, "P1: %d", game->players[0].score);
    draw_text(50, 20, buffer, COLOR_PLAYER1, FONT_NORMAL);

    // Puntuación Jugador 2
    sprintf(buffer, "P2: %d", game->players[1].score);
    draw_text(screen_width - 100, 20, buffer, COLOR_PLAYER2, FONT_NORMAL);

    // FPS
    sprintf(buffer, "FPS: %d", calculate_fps());
    draw_text(screen_width / 2, 20, buffer, COLOR_UI, FONT_SMALL);

    // Turbo meter
    draw_turbo_bar(&game->players[0], 50, screen_height - 30);
    draw_turbo_bar(&game->players[1], screen_width - 150, screen_height - 30);
}
```

2.4 Fase 4: Extensión a 3D

2.4.1 Preparación para 3D

2.4.1.1 Motor de Matemáticas 3D

Implementar estructuras y operaciones básicas:

```
// Vectores 3D
typedef struct {
    float x, y, z;
} vec3_t;

vec3_t vec3_add(vec3_t a, vec3_t b);
vec3_t vec3_sub(vec3_t a, vec3_t b);
vec3_t vec3_mul(vec3_t v, float scalar);
float vec3_dot(vec3_t a, vec3_t b);
vec3_t vec3_cross(vec3_t a, vec3_t b);
float vec3_length(vec3_t v);
vec3_t vec3_normalize(vec3_t v);

// Matrices 4x4
typedef struct {
    float m[4][4];
} mat4_t;

mat4_t mat4_identity(void);
mat4_t mat4_multiply(mat4_t a, mat4_t b);
vec3_t mat4_mul_vec3(mat4_t m, vec3_t v);
mat4_t mat4_translate(float x, float y, float z);
mat4_t mat4_rotate_x(float angle);
mat4_t mat4_rotate_y(float angle);
mat4_t mat4_rotate_z(float angle);
mat4_t mat4_scale(float x, float y, float z);
```

Importante: Todas estas operaciones deben implementarse sin usar librerías de matemáticas externas. Se pueden optimizar partes críticas en Assembly usando instrucciones SSE/AVX.

2.4.1.2 Proyección Perspectiva

Transformar coordenadas 3D a 2D para renderizado:

```
typedef struct {
    float fov;        // Field of view en grados
    float aspect;     // Aspect ratio (width/height)
    float near;       // Near clipping plane
    float far;        // Far clipping plane
} camera_t;

mat4_t mat4_perspective(camera_t* cam) {
    mat4_t result = {0};
```

```
    float f = 1.0f / tan(cam->fov * 0.5f * PI / 180.0f);

    result.m[0][0] = f / cam->aspect;
    result.m[1][1] = f;
    result.m[2][2] = (cam->far + cam->near) / (cam->near - cam->far);
    result.m[2][3] = -1.0f;
    result.m[3][2] = (2.0f * cam->far * cam->near) / (cam->near - cam->far);

    return result;
}

// Proyectar punto 3D a 2D
typedef struct { int x, y; } point2d_t;

point2d_t project_3d_to_2d(vec3_t point, mat4_t view, mat4_t proj) {
    // Aplicar transformaciones
    vec3_t clip = mat4_mul_vec3(proj, mat4_mul_vec3(view, point));

    // Perspective divide
    float w = clip.z;
    if (w == 0.0f) w = 0.0001f;

    float x_ndc = clip.x / w;
    float y_ndc = clip.y / w;

    // NDC to screen space
    point2d_t screen;
    screen.x = (x_ndc + 1.0f) * 0.5f * screen_width;
    screen.y = (1.0f - y_ndc) * 0.5f * screen_height;

    return screen;
}
```

2.4.1.3 Z-Buffer

Para ocultar superficies no visibles:

```
typedef struct {
    float* depth_buffer;  // Array de width * height
    uint16_t width;
    uint16_t height;
} zbuffer_t;

void zbuffer_init(zbuffer_t* zb, uint16_t w, uint16_t h) {
    zb->width = w;
    zb->height = h;
    zb->depth_buffer = kmalloc(w * h * sizeof(float));
}

void zbuffer_clear(zbuffer_t* zb) {
    for (int i = 0; i < zb->width * zb->height; i++) {
        zb->depth_buffer[i] = INFINITY;
    }
}
```

```
bool zbuffer_test(zbuffer_t* zb, int x, int y, float depth) {
    int idx = y * zb->width + x;
    if (depth < zb->depth_buffer[idx]) {
        zb->depth_buffer[idx] = depth;
        return true;
    }
    return false;
}
```

2.4.2 TRON 3D - Conceptos

2.4.2.1 Arena 3D

```
typedef struct {
    vec3_t position;
    vec3_t direction;
    vec3_t up;
    float speed;
    uint32_t color;
    bool alive;
} player3d_t;

typedef struct {
    vec3_t start;
    vec3_t end;
    float height;
    uint32_t color;
} wall_segment_t;

typedef struct {
    player3d_t players[2];
    wall_segment_t walls[MAX_WALLS];
    uint32_t wall_count;
    camera_t camera;
    vec3_t arena_size;  // Límites del mundo
} game3d_state_t;
```

2.4.2.2 Cámara

Opciones de cámara:

1. **Primera persona**: La cámara sigue al jugador
2. **Tercera persona**: Cámara detrás del jugador
3. **Aérea**: Vista desde arriba (híbrido 2D/3D)
4. **Espectador**: Cámara libre

```
void update_camera_third_person(camera_t* cam, player3d_t* player) {
    // Posicionar cámara detrás del jugador
    vec3_t offset = vec3_mul(player->direction, -10.0f);
    offset.y += 5.0f;  // Elevar la cámara
```

```
    cam->position = vec3_add(player->position, offset);
    cam->look_at = player->position;
}
```

2.4.2.3 Trails 3D

Los trails ahora son paredes verticales:

```
void create_wall_segment(game3d_state_t* game, player3d_t* player) {
    wall_segment_t wall;
    wall.start = player->last_position;
    wall.end = player->position;
    wall.height = 3.0f;  // Altura de la pared
    wall.color = player->color;

    game->walls[game->wall_count++] = wall;
}
```

2.4.2.4 Física Mejorada

```
typedef struct {
    vec3_t velocity;
    vec3_t acceleration;
    float friction;
    float turn_speed;
} physics3d_t;

void update_physics(player3d_t* player, physics3d_t* phys, float dt) {
    // Aplicar aceleración
    phys->velocity = vec3_add(phys->velocity, vec3_mul(phys->acceleration, dt));

    // Aplicar fricción
    phys->velocity = vec3_mul(phys->velocity, 1.0f - phys->friction * dt);

    // Actualizar posición
    player->position = vec3_add(player->position, vec3_mul(phys->velocity, dt));

    // Giros con inercia
    if (input_turn_left) {
        float angle = -phys->turn_speed * dt;
        player->direction = vec3_rotate_y(player->direction, angle);
    }
}
```

2.4.3 Renderizado 3D

2.4.3.1 Wireframe

El modo más simple, solo dibujar aristas:

```
void render_wireframe(game3d_state_t* game) {
    mat4_t view = mat4_look_at(game->camera.position, game->camera.look_at);
    mat4_t proj = mat4_perspective(&game->camera);

    // Renderizar paredes
    for (uint32_t i = 0; i < game->wall_count; i++) {
        wall_segment_t* wall = &game->walls[i];

        // 4 vértices de la pared (rectángulo vertical)
        vec3_t v0 = wall->start;
        vec3_t v1 = wall->end;
        vec3_t v2 = vec3_add(wall->end, (vec3_t){0, wall->height, 0});
        vec3_t v3 = vec3_add(wall->start, (vec3_t){0, wall->height, 0});

        // Proyectar a 2D
        point2d_t p0 = project_3d_to_2d(v0, view, proj);
        point2d_t p1 = project_3d_to_2d(v1, view, proj);
        point2d_t p2 = project_3d_to_2d(v2, view, proj);
        point2d_t p3 = project_3d_to_2d(v3, view, proj);

        // Dibujar aristas
        sys_draw_line(p0.x, p0.y, p1.x, p1.y, wall->color);
        sys_draw_line(p1.x, p1.y, p2.x, p2.y, wall->color);
        sys_draw_line(p2.x, p2.y, p3.x, p3.y, wall->color);
        sys_draw_line(p3.x, p3.y, p0.x, p0.y, wall->color);
    }
}
```

2.4.3.2 Filled Polygons (Rasterización)

Relleno de triángulos usando scanline algorithm:

```
void draw_triangle_filled(point2d_t p0, point2d_t p1, point2d_t p2,
                          float z0, float z1, float z2, uint32_t color, zbuffer_t* zb) {
    // Ordenar vértices por Y
    if (p0.y > p1.y) swap(&p0, &p1);
    if (p1.y > p2.y) swap(&p1, &p2);
    if (p0.y > p1.y) swap(&p0, &p1);

    // Scanline rasterization
    for (int y = p0.y; y <= p2.y; y++) {
        // Calcular X inicial y final para esta scanline
        float x_start, x_end;
        float z_start, z_end;

        // Interpolar...

        for (int x = x_start; x <= x_end; x++) {
            float z = lerp(z_start, z_end, (x - x_start) / (x_end - x_start));

            if (zbuffer_test(zb, x, y, z)) {
                sys_draw_pixel(x, y, color);
            }
        }
```

```
    }
}
```

2.4.3.3 Flat Shading

Iluminación básica por polígono:

```
typedef struct {
    vec3_t position;
    vec3_t direction;
    float intensity;
    uint32_t color;
} light_t;

uint32_t calculate_flat_shading(vec3_t normal, light_t* light, uint32_t base_color) {
    // Producto punto entre normal y dirección de luz
    float diffuse = max(0.0f, vec3_dot(normal, light->direction));

    // Aplicar intensidad
    diffuse *= light->intensity;

    // Modular color base
    uint8_t r = ((base_color >> 16) & 0xFF) * diffuse;
    uint8_t g = ((base_color >> 8) & 0xFF) * diffuse;
    uint8_t b = (base_color & 0xFF) * diffuse;

    return (r << 16) | (g << 8) | b;
}
```

2.4.4 Optimizaciones para 3D

2.4.4.1 Culling

```
// Backface culling
bool is_backface(vec3_t v0, vec3_t v1, vec3_t v2, vec3_t camera_pos) {
    vec3_t edge1 = vec3_sub(v1, v0);
    vec3_t edge2 = vec3_sub(v2, v0);
    vec3_t normal = vec3_cross(edge1, edge2);
    vec3_t to_camera = vec3_sub(camera_pos, v0);

    return vec3_dot(normal, to_camera) < 0;
}

// Frustum culling (simple)
bool is_in_frustum(vec3_t point, camera_t* cam) {
    // Simplified AABB test
    vec3_t to_point = vec3_sub(point, cam->position);
    float distance = vec3_length(to_point);

    return distance >= cam->near && distance <= cam->far;
}
```

2.4.4.2 Assembly Crítico

Optimizar rutinas de rasterización en Assembly:

```
; Función optimizada para copiar pixel con Z-test
; RDI = framebuffer, RSI = zbuffer, RDX = x, RCX = y, R8 = z, R9 = color
draw_pixel_z_asm:
    ; Calcular índice: y * width + x
    mov rax, rcx
    imul rax, [screen_width]
    add rax, rdx

    ; Z-test
    cvtsi2ss xmm0, r8
    movss xmm1, [rsi + rax*4]
    comiss xmm0, xmm1
    jae .reject

    ; Escribir Z
    movss [rsi + rax*4], xmm0

    ; Escribir color
    mov [rdi + rax*4], r9d

.reject:
    ret
```

3 Manual de Usuario

3.1 Introducción al Sistema ARES

ARES (ARES Recursive Experimental System) es un sistema operativo educativo diseñado para demostrar los conceptos fundamentales de arquitectura de computadoras y programación de sistemas en bajo nivel.

El sistema arranca directamente desde el hardware y proporciona una interfaz de línea de comandos (shell) desde la cual se pueden ejecutar diversos programas, siendo el juego TRON la aplicación principal.

3.2 Arranque del Sistema

3.2.1 En QEMU

Para arrancar ARES en el emulador QEMU:

```
qemu-system-x86_64 -drive format=raw,file=ares.img -m 256M
```

Opciones adicionales recomendadas:

```
# Con aceleración KVM (Linux)
qemu-system-x86_64 -enable-kvm -drive format=raw,file=ares.img -m 256M

# Con resolución específica
qemu-system-x86_64 -drive format=raw,file=ares.img -m 256M -vga std
```

3.2.2 En VirtualBox

1. Crear una nueva máquina virtual:
   - Tipo: Other
   - Versión: Other/Unknown (64-bit)
   - Memoria: 256MB (mínimo)
2. Configurar almacenamiento:
   - Añadir `ares.img` como disco duro
3. Configuración de video:
   - Controlador gráfico: VBoxVGA
   - Memoria de video: 16MB

3.2.3 En Hardware Real

Importante: Para ejecutar en hardware real, grabar la imagen en un dispositivo USB booteable:

```
# Linux/macOS
sudo dd if=ares.img of=/dev/sdX bs=4M status=progress

# Windows (usar Rufus o similar)
```

Luego configurar la BIOS/UEFI para arrancar desde USB en modo Legacy/CSM.

3.3 Uso de la Shell

3.3.1 Prompt

Al arrancar, el sistema mostrará el prompt:

```
ARES>
```

3.3.2 Comandos Disponibles

3.3.2.1 `help`

Muestra la lista de todos los comandos disponibles.

```
ARES> help
Comandos disponibles:
  help      - Muestra esta ayuda
  time      - Muestra la hora del sistema
  meminfo   - Información de memoria y registros
  cpuinfo   - Información del procesador
  clear     - Limpia la pantalla
  tron      - Inicia el juego TRON
  benchmark - Ejecuta benchmarks del sistema
```

3.3.2.2 `time`

Muestra la fecha y hora actual del sistema (obtenida del RTC).

```
ARES> time
Fecha: 18/10/2025
Hora: 14:32:15
```

3.3.2.3 `meminfo`

Muestra información sobre la memoria del sistema y los registros del procesador en el momento de la llamada.

```
ARES> meminfo
=== Información de Memoria ===
Memoria total: 256 MB
Memoria usada: 45 MB
```

```
Memoria libre: 211 MB

=== Registros del Procesador ===
RAX: 0x0000000000000001
RBX: 0x0000000000000000
RCX: 0x00000000000A0000
RDX: 0x0000000000000003
RSI: 0x0000000000100000
RDI: 0x0000000000200000
RBP: 0x000000000009FFF8
RSP: 0x000000000009FFE0
RIP: 0x0000000000101234
...
```

3.3.2.4 `cpuinfo`

Información del procesador detectado.

```
ARES> cpuinfo
Fabricante: GenuineIntel
Modelo: Intel(R) Core(TM) i7-9750H
Frecuencia: 2600 MHz
Cores detectados: 1
Características:
  - SSE: Sí
  - SSE2: Sí
  - AVX: Sí
  - x86-64: Sí
```

3.3.2.5 `clear`

Limpia la pantalla.

3.3.2.6 `tron`

Inicia el juego TRON.

```
ARES> tron
```

3.3.2.7 `benchmark`

Ejecuta una serie de benchmarks del sistema.

```
ARES> benchmark
=== Benchmarks ARES ===
Test 1: FPS (frames por segundo)
  Renderizando 1000 frames...
  Resultado: 58.3 FPS

Test 2: Operaciones de punto flotante
  Ejecutando 1M de operaciones...
  Resultado: 234 MFLOPS
```

```
Test 3: Acceso a memoria
  Leyendo 100MB...
  Resultado: 1234 MB/s

Test 4: Acceso a video
  Escribiendo framebuffer...
  Resultado: 456 MB/s
```

3.4 Juego TRON

3.4.1 Inicio del Juego

Desde la shell, ejecutar:

```
ARES> tron
```

Se mostrará el menú principal.

3.4.2 Menú Principal

```
╔═══════════════════════════╗
║         T R O N           ╚═══════════════════════════╗
╚═══════════════════════════╝

  1. Un Jugador
  2. Dos Jugadores
  3. Configuración
  4. Salir

Selecciona una opción:
```

3.4.3 Controles

3.4.3.1 Modo 2 Jugadores

Jugador 1 (Color Cian):

- `W`: Arriba
- `S`: Abajo
- `A`: Izquierda
- `D`: Derecha
- `Shift Izq`: Turbo (si está disponible)

Jugador 2 (Color Magenta):

- `Flecha Arriba`: Arriba
- `Flecha Abajo`: Abajo
- `Flecha Izq`: Izquierda
- `Flecha Der`: Derecha
- `Shift Der`: Turbo (si está disponible)

3.4.3.2 Controles Generales

- `ESC`: Pausar / Volver al menú
- `R`: Reiniciar partida
- `+`: Aumentar velocidad (en pausa)
- `-`: Disminuir velocidad (en pausa)

3.4.4 Configuración

En el menú de configuración se pueden ajustar:

```
╔═══════════════════════════════╗
║      CONFIGURACIÓN            ║
╚═══════════════════════════════╝

  Velocidad del juego: [Normal]
    < Lento | Normal | Rápido | Turbo >

  Obstáculos: [Activados]
    [ ] Desactivados
    [X] Activados

  Número de obstáculos: [5]
    [ 0 - 20 ]

  Sonido: [Activado]
    [X] Activado
    [ ] Desactivado

  Tamaño de pantalla: [Normal]
    < Pequeño | Normal | Grande >

  [Guardar]  [Cancelar]
```

3.4.5 Reglas del Juego

1. Cada jugador controla una «moto de luz» que deja un rastro detrás
2. Si chocas contra un rastro (tuyo o del oponente), pierdes
3. Si chocas contra los bordes de la pantalla, pierdes
4. Si chocas contra un obstáculo, pierdes
5. El último jugador en pie gana la ronda
6. El primer jugador en ganar 3 rondas gana la partida

3.4.6 Interfaz del Juego

Durante una partida se muestra:

```
┌──────────────────────────────────────────────┐
│ P1: 2  ▓▓▓▓▓▓▒▒  TURBO      FPS: 60      P2: 1│
├──────────────────────────────────────────────┤
│                                              │
│                                              │
│     [Área de juego con grid y trails]          │
```

```
│                                                    │
│                                                    │
└────────────────────────────────────────────────────┘
```

- Esquina superior izquierda: Puntuación Jugador 1 y barra de turbo
- Centro superior: FPS actual
- Esquina superior derecha: Puntuación Jugador 2 y barra de turbo

3.4.7 Pantalla de Game Over

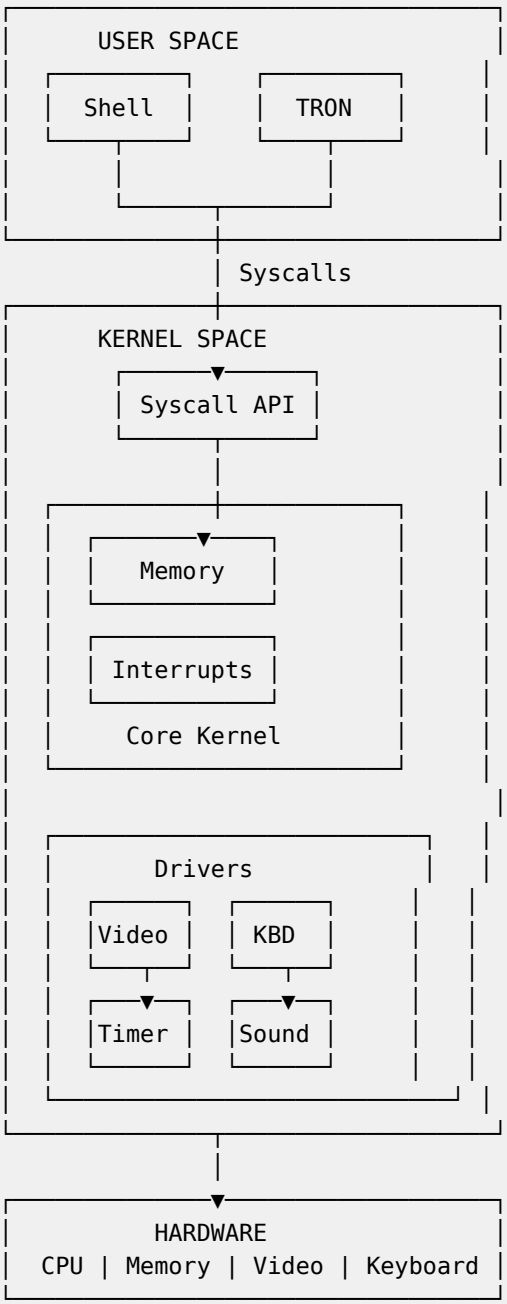
```
╔═══════════════════════════════╗
║         GAME OVER             ║
╚═══════════════════════════════╝

      Jugador 2 Gana!

  Puntuación Final:
    Jugador 1: 2
    Jugador 2: 3

  [R] Jugar de nuevo
  [M] Volver al menú
  [Q] Salir
```

4 Informe Técnico del Diseño

4.1 Arquitectura del Sistema

4.1.1 Modelo de Capas

El sistema ARES sigue un modelo de capas típico de sistemas operativos:



4.1.2 Convenciones de Llamada

El sistema utiliza la System V AMD64 ABI para llamadas entre C y Assembly:

Pasaje de argumentos (enteros/punteros):

1. RDI
2. RSI
3. RDX
4. RCX
5. R8
6. R9

7+ Stack

Pasaje de argumentos (punto flotante): 1-8. XMM0-XMM7

Valor de retorno:

- Enteros: RAX (RDX para valores de 128 bits)
- Flotantes: XMM0

Registros preservados (callee-saved):

- RBX, RBP, R12-R15

Registros volátiles (caller-saved):

- RAX, RCX, RDX, RSI, RDI, R8-R11

4.1.3 Mapa de Memoria

```
0x0000000000000000 - 0x0000000000000FFF  : Null page (no mapeado)
0x0000000000001000 - 0x0000000000001FFF  : GDT
0x0000000000002000 - 0x0000000000002FFF  : IDT
0x0000000000003000 - 0x0000000000003FFF  : Page tables
0x0000000000004000 - 0x0000000000004FFF  : E820 Memory map
0x0000000000005000 - 0x0000000000005FFF  : System variables
0x0000000000006000 - 0x000000000006FFFF  : Bootloader code
0x0000000000070000 - 0x000000000007FFFF  : AHCI tables
0x0000000000080000 - 0x000000000009FFFF  : Kernel stack
0x0000000000100000 - 0x00000000003FFFFF  : Kernel code + data
0x0000000000400000 - 0x00000000007FFFFF  : Heap del kernel
0x0000000000800000 - 0x0000000000FFFFFF  : User space
0x00000000FD000000 - 0x00000000FDFFFFFF  : Video framebuffer (mapeado)
```

4.2 Diseño de Drivers

4.2.1 Driver de Video

4.2.1.1 Inicialización

El driver de video se inicializa en varias etapas:

1. **Bootloader**: Detección VESA y configuración de modo gráfico
2. **Kernel**: Mapeo del framebuffer a memoria virtual
3. **Runtime**: Inicialización de double buffering

```
```

```c
void video_init(void) {
    // Obtener info VESA del bootloader
    vbe_info_t* vbe = (vbe_info_t*)VBEModeInfoBlock;

    // Configurar estructura global
    video.framebuffer = (uint32_t*)vbe->PhysBasePtr;
    video.width = vbe->XResolution;
    video.height = vbe->YResolution;
    video.bpp = vbe->BitsPerPixel;
    video.pitch = vbe->BytesPerScanLine;

    // Allocar back buffer
    size_t fb_size = video.pitch * video.height;
    video.back_buffer = kmalloc(fb_size);

    // Limpiar pantallas
    video_clear(COLOR_BLACK);
    memset(video.back_buffer, 0, fb_size);
}
```

4.2.1.2 Primitivas Gráficas

Put Pixel (función más crítica):

```c
void video_put_pixel(uint16_t x, uint16_t y, uint32_t color) {
    if (x >= video.width || y >= video.height) return;

    uint32_t* pixel = video.back_buffer + y * (video.pitch / 4) + x;
    *pixel = color;
}
```

Draw Line (Bresenham):

```c
void video_draw_line(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint32_t
color) {
    int dx = abs(x1 - x0);
    int dy = abs(y1 - y0);
    int sx = x0 < x1 ? 1 : -1;
    int sy = y0 < y1 ? 1 : -1;
    int err = dx - dy;

    while (1) {
        video_put_pixel(x0, y0, color);

        if (x0 == x1 && y0 == y1) break;

        int e2 = 2 * err;
        if (e2 > -dy) {
            err -= dy;
            x0 += sx;
        }
        if (e2 < dx) {
            err += dx;
```

```
            y0 += sy;
        }
    }
}
```

4.2.2 Driver de Teclado

4.2.2.1 Tabla de Scan Codes

```
static const char scancode_to_ascii[128] = {
    0,  27, '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '=', '\b',
    '\t', 'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', '[', ']', '\n',
    0,    'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', ';', '\'', '`',
    0,    '\\', 'z', 'x', 'c', 'v', 'b', 'n', 'm', ',', '.', '/',
    // ... resto de la tabla
};
```

4.2.2.2 Handler de Interrupción

```
; keyboard_irq_handler.asm
global keyboard_irq_handler
extern keyboard_handle_scancode

keyboard_irq_handler:
    push rax
    push rbx
    push rcx
    push rdx

    ; Leer scan code del puerto 0x60
    in al, 0x60

    ; Llamar handler en C
    movzx rdi, al
    call keyboard_handle_scancode

    ; Enviar EOI al PIC
    mov al, 0x20
    out 0x20, al

    pop rdx
    pop rcx
    pop rbx
    pop rax
    iretq
```

```
void keyboard_handle_scancode(uint8_t scancode) {
    // Scancode de release (bit 7 = 1)
    if (scancode & 0x80) {
        scancode &= 0x7F;
```

```
        keyboard.keys_pressed[scancode] = 0;
        return;
    }

    // Scancode de press
    keyboard.keys_pressed[scancode] = 1;

    // Agregar al buffer
    char c = scancode_to_ascii[scancode];
    if (c != 0) {
        keyboard.buffer[keyboard.write_pos] = c;
        keyboard.write_pos = (keyboard.write_pos + 1) % 256;
    }
}
```

4.3 Gestión de Excepciones

4.3.1 Implementación

Cada excepción tiene un handler específico:

```
void exception_division_by_zero(interrupt_frame_t* frame) {
    printf("\n*** EXCEPTION: Division by Zero ***\n");
    print_exception_info(frame);
    halt_system();
}

void exception_invalid_opcode(interrupt_frame_t* frame) {
    printf("\n*** EXCEPTION: Invalid Opcode ***\n");
    print_exception_info(frame);

    // Mostrar bytes alrededor de RIP
    printf("Code dump at RIP:\n");
    dump_memory((void*)frame->rip, 32);

    halt_system();
}

void exception_general_protection(interrupt_frame_t* frame, uint64_t error_code) {
    printf("\n*** EXCEPTION: General Protection Fault ***\n");
    printf("Error code: 0x%llX\n", error_code);

    if (error_code & 0x1) printf("  - External event\n");
    if (error_code & 0x2) printf("  - IDT reference\n");
    else if (error_code & 0x4) printf("  - LDT reference\n");
    else printf("  - GDT reference\n");

    printf("Selector index: %llu\n", (error_code >> 3) & 0x1FFF);

    print_exception_info(frame);
    halt_system();
}
```

```
void print_exception_info(interrupt_frame_t* frame) {
    printf("Registers:\n");
    printf("  RAX: 0x%016llX  RBX: 0x%016llX\n", frame->rax, frame->rbx);
    printf("  RCX: 0x%016llX  RDX: 0x%016llX\n", frame->rcx, frame->rdx);
    printf("  RSI: 0x%016llX  RDI: 0x%016llX\n", frame->rsi, frame->rdi);
    printf("  RBP: 0x%016llX  RSP: 0x%016llX\n", frame->rbp, frame->rsp);
    printf("  R8:  0x%016llX  R9:  0x%016llX\n", frame->r8, frame->r9);
    printf("  R10: 0x%016llX  R11: 0x%016llX\n", frame->r10, frame->r11);
    printf("  R12: 0x%016llX  R13: 0x%016llX\n", frame->r12, frame->r13);
    printf("  R14: 0x%016llX  R15: 0x%016llX\n", frame->r14, frame->r15);
    printf("\n");
    printf("  RIP: 0x%016llX\n", frame->rip);
    printf("  RFLAGS: 0x%016llX\n", frame->rflags);
    printf("  CS: 0x%04X  SS: 0x%04X\n", frame->cs, frame->ss);
}
```

4.3.2 Recuperación

En lugar de colgar el sistema, intentamos retornar a la shell:

```
void recover_from_exception(void) {
    // Limpiar estado
    clear_keyboard_buffer();

    // Mensaje
    printf("\nPresiona cualquier tecla para volver a la shell...\n");
    keyboard_get_char();

    // Saltar de vuelta a la shell
    longjmp(shell_recovery_point, 1);
}
```

4.4 Benchmarking

4.4.1 FPS Counter

```
typedef struct {
    uint32_t frame_count;
    uint64_t last_time;
    float current_fps;
} fps_counter_t;

void fps_update(fps_counter_t* fps) {
    fps->frame_count++;

    uint64_t now = sys_get_ticks();
    uint64_t elapsed = now - fps->last_time;

    // Actualizar cada segundo
    if (elapsed >= 1000) {
```

```
        fps->current_fps = (float)fps->frame_count * 1000.0f / elapsed;
        fps->frame_count = 0;
        fps->last_time = now;
    }
}
```

4.4.2 Benchmark de Floating Point

```
float benchmark_floating_point(void) {
    uint64_t start = sys_get_ticks();

    volatile float result = 0.0f;
    const uint32_t iterations = 1000000;

    for (uint32_t i = 0; i < iterations; i++) {
        result += sqrtf(i * 3.14159f);
        result *= 1.00001f;
        result /= 1.00002f;
    }

    uint64_t end = sys_get_ticks();
    uint64_t elapsed_ms = end - start;

    // MFLOPS = (operaciones * iteraciones) / (tiempo_en_segundos * 1M)
    float mflops = (4.0f * iterations) / (elapsed_ms / 1000.0f) / 1000000.0f;

    return mflops;
}
```

4.5 Consideraciones para Extensión a 3D

4.5.1 Performance

Para lograr renderizado 3D en tiempo real:

1. **Target**: Mínimo 30 FPS para experiencia jugable
2. **Resolución**: Reducir a 640×480 o similar para mayor performance
3. **Polígonos**: Limitar complejidad de escena (1000-5000 triángulos)
4. **Optimización**: Código crítico en Assembly con SSE/AVX

Tip: Implementar un sistema de niveles de detalle (LOD): renderizar objetos lejanos con menos polígonos.

4.5.2 Arquitectura Modular

Separar el motor 3D en módulos independientes:

```
tron3d/
├── math/          # Vectores, matrices, quaternions
```

```
├── renderer/      # Pipeline de renderizado
│   ├── vertex.c   # Transform & lighting
│   ├── raster.c   # Rasterización
│   └── zbuffer.c  # Depth testing
├── physics/       # Física y colisiones
├── camera/        # Sistema de cámara
└── game/          # Lógica específica de TRON
```

4.5.3 Testing Incremental

Desarrollar en etapas:

1. Renderizar cubo wireframe rotando
2. Agregar rasterización de polígonos
3. Implementar Z-buffer
4. Agregar iluminación básica
5. Integrar con física del juego
6. Optimizar

4.6 Conclusiones

El proyecto ARES representa una implementación completa de un sistema operativo educativo desde cero, demostrando:

- Programación de bajo nivel en x86-64
- Gestión de hardware sin abstracciones
- Diseño de APIs y interfaces
- Optimización de rendimiento
- Arquitectura extensible

La implementación del juego TRON sirve como vehículo perfecto para demostrar capacidades gráficas, manejo de entrada, física simple y la integración de todos los componentes del sistema.

La extensión planificada a 3D presenta desafíos adicionales interesantes en matemáticas, renderizado y optimización, preparando el terreno para futuras expansiones hacia un sistema operativo más completo.

Importante: Este documento debe ser actualizado conforme se implementan las diferentes fases del proyecto, documentando decisiones de diseño, problemas encontrados y soluciones implementadas.

4.7 Referencias

4.7.1 Documentación Técnica

- Intel 64 and IA-32 Architectures Software Developer's Manual
- System V Application Binary Interface AMD64 Architecture
- OSDev Wiki: https://wiki.osdev.org
- VESA BIOS Extension (VBE) Core Functions Standard
- Pure64 Bootloader Documentation

4.7.2 Algoritmos y Técnicas

- Bresenham's Line Algorithm
- Scanline Rasterization
- Z-Buffer Algorithm
- Perspective Projection Matrices

4.7.3 Inspiración

- TRON: Ares (2025) - Inspiración del nombre del proyecto
- Armagetron Advanced - Implementación moderna de TRON 3D