Proyecto 1
Computación Paralela y Distribuida

Hugo Rivas - 22500
Alexis Mesias - 22562
Mauricio Lemus - 22461

Link al video: https://youtu.be/lcaA8iTPbeI
Link al github: https://github.com/Riv2oo4/Proyecto1_Paralela.git

Introducción

El presente documento describe el desarrollo, análisis y documentación de un programa escrito en C que implementa una simulación gráfica de bolas en movimiento parabólico con colisiones, efectos visuales y partículas. El código aprovecha programación paralela mediante OpenMP para mejorar la eficiencia en la actualización de los objetos simulados.
La importancia de este proyecto radica en la integración de procesamiento concurrente con técnicas de renderizado en tiempo real, lo cual constituye un ejemplo práctico de cómo la programación paralela puede aplicarse en aplicaciones gráficas y simulaciones físicas.

Antecedentes

La simulación física de partículas y objetos ha sido un área recurrente de estudio tanto en la investigación académica como en la industria del entretenimiento (videojuegos, gráficos por computadora y realidad virtual). Tradicionalmente, estos cálculos se realizan de forma secuencial, lo cual limita la escalabilidad y el rendimiento.

Con la inclusión de librerías como OpenMP, es posible paralelizar secciones críticas, distribuyendo las cargas de trabajo en múltiples hilos de ejecución. Según Chapman et al. (2008), *"OpenMP proporciona un modelo de programación paralela sencillo y efectivo que permite a los desarrolladores aprovechar arquitecturas multiprocesador sin reescribir completamente sus aplicaciones"*.

En este contexto, el presente programa implementa:

- Actualización concurrente de bolas y partículas.
- Mecanismos de sincronización mediante regiones críticas.
- Dibujo con doble buffer en entorno Win32 para evitar parpadeos (flicker).

Cuerpo del Informe

Diseño del programa

El programa sigue la arquitectura de una aplicación Win32 con bucle principal:

1. Inicialización: creación de ventana, backbuffer, variables globales y memoria dinámica para las bolas.
2. Captura de argumentos: la función ParseN recibe y valida el número de bolas a simular desde la línea de comandos.
3. Ciclo principal: maneja mensajes del sistema y ejecuta las fases de simulación (física + renderizado).
4. Finalización: liberación de memoria y objetos gráficos (GDI).

Programación defensiva

El programa incorpora múltiples medidas defensivas:

- Validación de N para evitar valores negativos o mayores al límite (MAX_N).
- Uso de clampi y clampf para asegurar rangos en valores físicos.
- Liberación segura de recursos GDI (FreeBalls).

Secciones paralelas

Las secciones críticas se distribuyen con #pragma omp parallel for en:

- Bolas: actualización de posiciones, colisiones y propiedades físicas.
- Partículas: cálculo de trayectorias y rebotes.

Mecanismos de sincronía

Para evitar condiciones de carrera:

- #pragma omp critical(rng) protege el uso de rand() en entornos multihilo.
- #pragma omp critical(sparks) asegura la asignación correcta de slots en el pool de partículas.

Resultados

La simulación logra mantener un frame rate constante gracias al control de tiempo (FRAME_MS) y la utilización de buffers gráficos. Además, el uso de paralelismo permite una escalabilidad significativa al aumentar el número de bolas simuladas.

Citas textuales / Píe de página

- *"OpenMP ofrece un modelo flexible de programación paralela de memoria compartida que facilita el desarrollo de aplicaciones eficientes"* (Dagum & Menon, 1998).
- *"La técnica de double buffering es fundamental para evitar el parpadeo en aplicaciones gráficas interactivas"*(Foley et al., 1995).
- *"La programación defensiva busca anticipar posibles errores o entradas no válidas para asegurar la estabilidad del software"* (McConnell, 2004).

Conclusiones y Recomendaciones

- La implementación del programa demuestra que el uso de paralelismo con OpenMP mejora la eficiencia en simulaciones gráficas de múltiples entidades.
- El empleo de programación defensiva garantiza estabilidad y robustez frente a entradas inválidas o estados no previstos.
- Se recomienda, para futuras mejoras:
    1. Migrar la parte gráfica a librerías multiplataforma como SDL2 o OpenGL, facilitando portabilidad a sistemas Unix/Mac.
    2. Implementar balanceo dinámico de carga en OpenMP (schedule(dynamic)) para escenarios con más partículas.
    3. Incorporar test unitarios en funciones auxiliares (clampi, rand_inclusive_safe) para validar comportamientos.

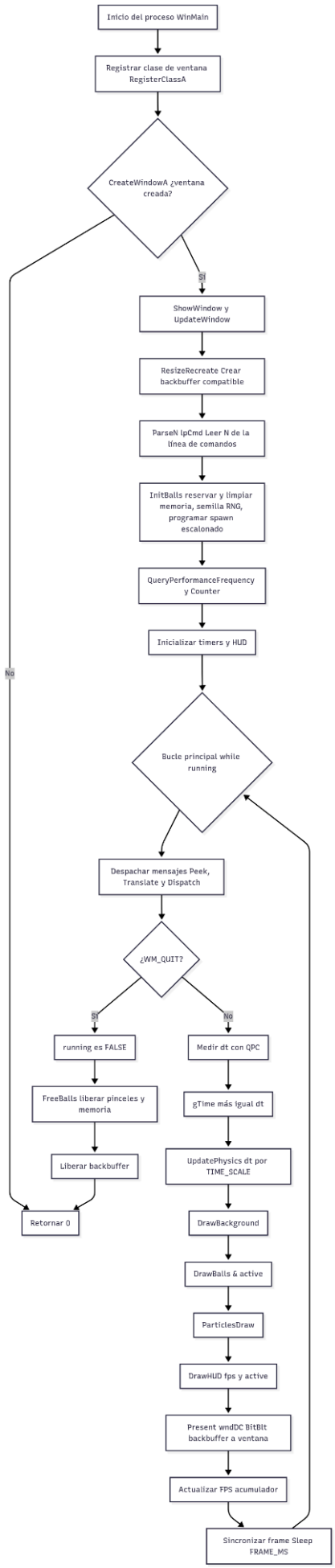
Apéndice

- Anexo 1: Diagrama de flujo del programa.
- Anexo 2: Catálogo de funciones (tabla).
- Archivos suplementarios: código fuente completo, capturas de pantalla de la simulación y logs de rendimiento.

Bibliografía

1. Dagum, L., & Menon, R. (1998). *OpenMP: An industry standard API for shared-memory programming*. IEEE Computational Science and Engineering, 5(1), 46–55.
2. Chapman, B., Jost, G., & van der Pas, R. (2008). *Using OpenMP: Portable Shared Memory Parallel Programming*. MIT Press.
3. Foley, J., van Dam, A., Feiner, S., & Hughes, J. (1995). *Computer Graphics: Principles and Practice*. Addison-Wesley.
4. McConnell, S. (2004). *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press.

Anexo 1


Inicio del proceso WinMain
Registrar clase de ventana RegisterClassA
CreateWindowA ¿ventana creada?
No
Sí
ShowWindow y UpdateWindow
ResizeRecreate Crear backbuffer compatible
ParseN lpCmd Leer N de la línea de comandos
InitBalls reservar y limpiar memoria, semilla RNG, programar spawn escalonado
QueryPerformanceFrequency y Counter
Inicializar timers y HUD
Bucle principal while running
Despachar mensajes Peek, Translate y Dispatch
¿WM_QUIT?
Sí
No
running es FALSE
FreeBalls liberar pinceles y memoria
Liberar backbuffer
Retornar 0
Medir dt con QPC
gTime más igual dt
UpdatePhysics dt por TIME_SCALE
DrawBackground
DrawBalls & active
ParticlesDraw
DrawHUD fps y active
Present wndDC BitBlt backbuffer a ventana
Actualizar FPS acumulador
Sincronizar frame Sleep FRAME_MS

Anexo 2

Función / Estructura	Entradas (nombre, tipo, uso)	Salidas (nombre, tipo, uso)	Descripción (propósito y funcionamiento)
Ball (struct)	x,y,vx,vy (float)posición/velocidades; r (int) radio; color (COLORREF); brush,shadow (HBRUSH) pinceles; active (BOOL)estado; animación (angle, angVel, squash…)	*(no aplica)*	Representa una bola con estado físico, color, pinceles y estela opcional. Elemento principal de la simulación.
Particle (struct)	x,y,vx,vy (float)posición/velocidades; life,maxLife (float) duración; size (int)tamaño; brush (HBRUSH) color; alive (BOOL) estado	*(no aplica)*	Partícula efímera usada para chispas/efectos. Gestionadas en pool global.
Darken(c:int,pct:int)	c (COLORREF), pct (int, % oscurecer)	COLORREFcolor oscurecido	Devuelve el color más oscuro reduciendo RGB proporcionalmente.
GroundY()	*(ninguna, usa height,floorHglobales)*	floatcoordenada Y del piso	Calcula el nivel del suelo según alto de ventana y altura de piso.
clampi(v,a,b) / clampf(v,a,b)	v (int/float), a min, b max	Valor limitado en [a,b]	Funciones defensivas para mantener valores dentro de un rango.
rand_inclusive_safe(lo:int, hi:int)	lo,hi (int)	intaleatorio entre lo y hi	RNG protegido con omp critical(rng). Evita condiciones de carrera.
NextIntervalSafe()	*(ninguna)*	doubleintervalo aleatorio ~0.12–0.22 s	Igual que NextInterval pero protegido con critical(rng).
NextInterval()	*(ninguna)*	doubleintervalo aleatorio	Versión no protegida, solo en contexto serial.
FreeBalls()	*(ninguna, usa globals)*	*(ninguna)*	Libera memoria de balls y pinceles GDI asociados. Defensiva: chequea nulos.
MakeBrushes(b:Ball*)	b (Ball*)	*(ninguna)*	Crea pinceles de color y sombra para la bola. Reemplaza los previos si existen.
ParticlesClear()	*(ninguna)*	*(ninguna)*	Reinicia pool de partículas: todas alive=FALSE.
SpawnSparks(x,y:float, count:int, brush:HBRUSH, baseVx:float)	Posición, cantidad, pincel opcional, velocidad base	*(ninguna)*	Genera partículas de chispa. Protege acceso al pool con critical(sparks).

ParticlesUpdate(dt:double)	dt (tiempo)	*(ninguna)*	Actualiza partículas en paralelo (omp parallel for): vida, posición, rebote en suelo.
ParticlesDraw()	*(ninguna)*	*(ninguna)*	Dibuja partículas activas en backDC. Serial (GDI no es thread-safe).
ActivateBall(b:Ball*)	b (Ball*)	*(ninguna)*	Inicializa una bola: posición, velocidad, color aleatorio, pinceles y animación.
ScheduleBallSerial(b:Ball*)	b (Ball*)	*(ninguna)*	Desactiva bola y programa reaparición con NextInterval (versión serial).
ScheduleBallSafe(b:Ball*, gy:float)	b (Ball*), gy (float piso)	*(ninguna)*	Igual que la anterior pero usa NextIntervalSafe. Diseñada para ejecución paralela.
InitBalls()	*(ninguna)*	*(ninguna)*	Reserva memoria para balls, crea gPortalBrush, inicializa spawn escalonado y limpia partículas.
InitBackBuffer(wndDC:HDC, w:int, h:int)	wndDC (HDC), w,h (int)	*(ninguna)*	Crea DC compatible + bitmap para backbuffer. Reemplaza si ya existía.
DrawBackground()	*(ninguna)*	*(ninguna)*	Dibuja gradiente de fondo y rectángulo de piso en backDC.
DrawBallWithEffects(b:Ball*)	b (Ball*)	*(ninguna)*	Dibuja sombra, bola deformada (squash) y brillo según ángulo.
DrawTrails(b:Ball*)	b (Ball*)	*(ninguna)*	Dibuja estela de la bola (si está habilitado ENABLE_TRAILS).
DrawBalls(outActive:int*)	Puntero a contador opcional	outActiveactualizado (int)	Dibuja todas las bolas activas, retorna cantidad en *outActive.
DrawHUD(fps:double, active:int)	FPS y bolas activas	*(ninguna)*	Escribe texto en backbuffer con FPS y número de bolas activas.
Present(wndDC:HDC)	wndDC (HDC ventana)	*(ninguna)*	Copia el backbuffer a ventana (BitBlt).
UpdatePhysics(dt:double)	dt (delta tiempo)	*(ninguna)*	Física principal: activa bolas pendientes, actualiza en paralelo posiciones, colisiones, squash, jitter y scheduling. Llama también a ParticlesUpdate.

ResizeRecreate()	*(ninguna)*	*(ninguna)*	Actualiza width,heightdesde hwnd y recrea backbuffer.
WndProc(h,msg,wParam,lParam)	Parámetros de ventana estándar	LRESULT	Maneja mensajes Win32 (WM_SIZE, WM_PAINT, WM_DESTROY).
ParseN(lpCmdLine:LPSTR)	lpCmdLine (cadena argumentos)	intnúmero de bolas	Parsea N defensivamente: si inválido → DEF_N, si mayor a MAX_N → lo limita.
WinMain(...)	Parámetros estándar Win32 (hInst,hPrev,lpCmd,nShow)	int código retorno	Programa principal: registra ventana, crea HWND, inicializa (ParseN, InitBalls), ejecuta bucle principal (mensajes, física, dibujo, FPS), y limpia al salir.

Anexo 3