


# CONSTRUIR, SIMULAR, PUBLICAR

Construcción y simulación física de vehículos 2D

Computación Gráfica CS4016 — UTEC

# UN JUEGO, CUATRO CAPAS

- El jugador **construye una combi** sobre una grilla, la prueba en terrenos irregulares y debe llegar a la meta.
- Referente conceptual: el ciclo *construir* → *simular* → *reintentar* de *Bad Piggies*, con implementación propia.
- C++23 + raylib; motor de rigid bodies, terreno y evaluación hechos por el equipo.




**01 MOTOR DE FÍSICAS**
Alejandro Ore

**02 GEOMETRÍA**
Kiara Balcazar

**03 DEPLOYMENT**
Juan Diego Castro

**04 GAMEPLAY**
Kayla De Vivanco

# 01 MOTOR DE FÍSICAS

Rigid bodies, colisiones, impulsos y rotura estructural, sin librería de física externa.

# RIGID BODIES Y PASO TEMPORAL

- Cada cuerpo guarda $x,\ \theta,\ v,\ \omega,\ m,\ m^{-1}$ e $I$.
- Formas: `Box`, `Circle`, `Capsule`, `Triangle`, `Edge` y `TerrainMesh` (estático: $m^{-1} = I^{-1} = 0$).

$$I_{\text{circle}} = \tfrac{1}{2} m r^2 \qquad I_{\text{box}} = \tfrac{1}{12} m (w^2 + h^2)$$

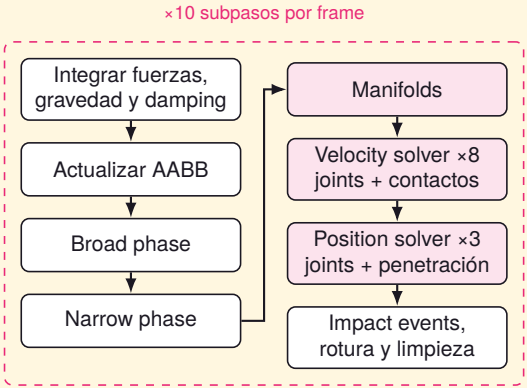
$$v_{t+\Delta t} = v_t + (F/m + g)\,\Delta t$$

$$x_{t+\Delta t} = x_t + v_{t+\Delta t}\,\Delta t, \quad \theta_{t+\Delta t} = \theta_t + \omega_{t+\Delta t}\,\Delta t$$

Euler semi-implícito: primero velocidad, luego posición.

| Parámetro | Valor |
|---|---|
| Gravedad | $(0,\ 392) = 9.8 \times 40$ px/s$^2$ |
| Subpasos por frame | 10 |
| Iteraciones de velocidad | 8 |
| Iteraciones de posición | 3 |
| Damping lineal / angular | 0.05 / 0.65 (exponencial) |
| Fricción / restitución | 1.25 / 0.2 |
| Protección 1.er aterrizaje | 0.35 s |
| Umbral de rotura | $30\times$ resistencia |

# PIPELINE DE UN PASO: WORLD::STEP

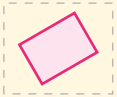



- Dividir el frame en subpasos reduce errores con piezas rápidas y cuerpos unidos.
- El **manifold** conecta detección y respuesta: se crea en la narrow phase y se reutiliza en ambos solvers y en los eventos de impacto.
- Cuerpos unidos por un joint no colisionan entre sí (caché de conectividad).

# DETECCIÓN: AABB, SAT Y CLIPPING

## BROAD PHASE

Descarta pares cuyas AABB no se solapan. Un solape aún no es colisión.



forma rotada y su AABB

## NARROW PHASE: SAT

Proyecta ambos polígonos en las normales de sus aristas. Si un eje separa, no hay colisión; si no, el menor *overlap* da la normal y profundidad.



## CLIPPING

La incident edge se recorta contra los planos de la reference face (Sutherland–Hodgman) → hasta 2 puntos de contacto estables.




se conservan puntos detrás

Dispatch especializado por par de formas (`CircleOnCircle`, `BoxOnBox`, `CapsuleOnTerrainMesh`, …). El terreno es una malla de triángulos: cada triángulo entra como unidad estática.

# RESPUESTA POR IMPULSOS SECUENCIALES

Velocidad de un punto de contacto con brazo $r$:

$$v_p = v + \omega\,(-r_y,\ r_x)$$

Impulso normal:

$$j_n = \frac{-(1+e)\,v_{rel}\cdot n}{m_A^{-1} + m_B^{-1} + K_{rot}}$$

$$K_{rot} = I_A^{-1}(r_A \times n)^2 + I_B^{-1}(r_B \times n)^2$$

- Un impacto lejos del centro de masa produce poca traslación y mucha rotación.
- **Fricción**: impulso tangente limitado por $\mu j_n$.
- Se acumulan impulsos y se re-resuelven los contactos con error (estilo Box2D).
- **Baumgarte**: velocidad correctiva proporcional a la penetración, acotada para no “explotar”.

# JOINTS: QUÉ UNE Y QUÉ SE ROMPE

### REVOLUTE (RUEDAS)

Los anchors en world space deben coincidir; la rotación relativa queda libre.

$$C_{pos} = a_B - a_A$$

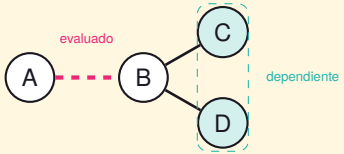
### WELD (ESTRUCTURA)

Además bloquea el ángulo relativo.

$$C_{angle} = \mathrm{wrap}(\theta_B - \theta_A - \theta_{relative})$$

Un joint se marca para rotura cuando su esfuerzo local supera $30\times$ su resistencia.

### JOINT GRAPH Y BRIDGE JOINTS

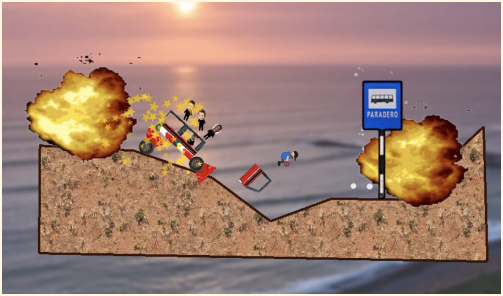



1. Excluir el joint; BFS desde cada extremo.
2. Componentes distintas → **puente**: carga la masa del lado dependiente.
3. Misma componente → redundante: se amortigua.

# EXPLOSIONES Y ESTABILIDAD

- El TNT aplica impulsos radiales con **atenuación lineal** según la distancia.
- Puede romper joints afectados y activar otro TNT en cadena.
- La limpieza elimina el joint, invalida la caché de conectividad y emite un evento visual.
- Rotura **local**: un joint sostiene lo que de verdad cuelga de él, no toda la combi.

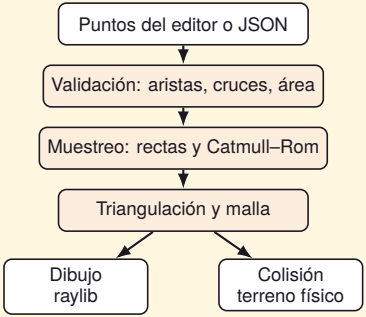
Límite honesto: solver iterativo pensado para la escala del juego, no para exactitud física en todos los regímenes.



# 02 GEOMETRÍA COMPUTACIONAL

Un mismo contorno se valida, se muestrea, se triangula y alimenta dibujo y colisión.

# DEL PUNTO A LA MALLA




- **Una sola ruta**: lo que el jugador ve es lo que toca la rueda.
- Celda $\leftrightarrow$ mundo:

$$p_{world} = o + s\,c, \qquad c = \left\lfloor \frac{p_{world} - o}{s} \right\rfloor$$

- Local $\rightarrow$ mundo (texturas y anchors):

$$p_{world} = x + R(\theta)\,q$$

La inversa usa $R(\theta)^T$.

# Validación del contorno

Un contorno es válido si:

- tiene $n \geq 3$ vértices;
- ninguna arista mide menos de $0.5$ u;
- no hay cruces entre aristas no adyacentes (orientación por producto cruzado, incluye puntos sobre una arista);
- $|A| \geq 1$ u$^2$, con $A = \frac{1}{2}\sum_i (x_i y_{i+1} - x_{i+1} y_i)$.

Costo directo $O(n^2)$, a cambio de diagnósticos visuales concretos.

| Caso | Vért. | Diagnóstico | Triáng. |
|---|---|---|---|
| Cuadrado | 4 | válido | 2 |
| Cóncavo | 5 | válido | 3 |
| Cruce | 4 | autointersección | 0 |
| Toque | 5 | autointersección | 0 |
| Repetido | 4 | arista corta | 0 |
| Colineal | 3 | área pequeña | 0 |

Casos de `proofs/terrain_geometry_proof.cpp`



rojo = problema; sigue editable

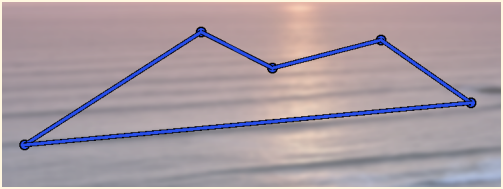
# CURVAS CATMULL-ROM ADAPTATIVAS

Tramo entre $p_1$ y $p_2$ (`Spline.cpp`):

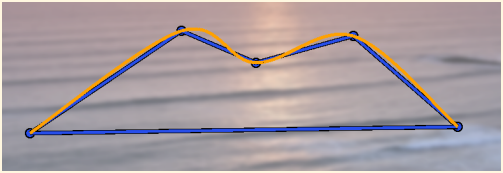
$$S(t) = \tfrac{1}{2}\big[2p_1 + (-p_0 + p_2)\,t + (2p_0 - 5p_1 + 4p_2 - p_3)\,t^2 + (-p_0 + 3p_1 - 3p_2 + p_3)\,t^3\big]$$

## MUESTREO ADAPTATIVO

1. Evaluar en $t = 0.25,\ 0.5,\ 0.75$.
2. Medir el error respecto a la cuerda *ab*.
3. Si supera la tolerancia $\varepsilon$ → dividir y recursar (máx. 12 niveles).
4. Si no → agregar *b*.



puntos de control y contorno lineal

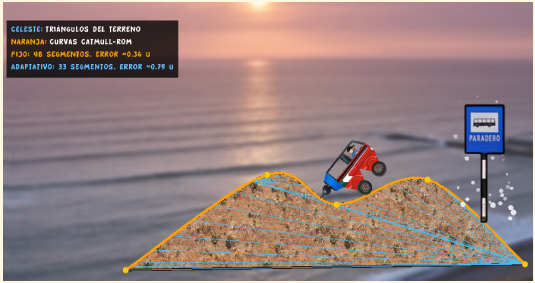


curva Catmull–Rom (naranja) sobre el contorno

# TRIANGULACIÓN POR RECORTE DE OREJAS

1. Orientar los índices en sentido antihorario.
2. Buscar un vértice **convexo** cuyo triángulo no contenga otros vértices.
3. Recortar esa oreja y repetir hasta quedar con 3.
4. Verificar: exactamente $n - 2$ triángulos, orientación positiva y $\sum A_{tri} = A_{\text{polígono}}$.

La misma lista de índices se usa para dibujar y para colisionar.




Modo gráfico: triángulos del terreno (celeste) y curva Catmull–Rom (naranja)

# MENOS GEOMETRÍA, MENOS TRABAJO PARA EL MOTOR

**-41.4%**
tiempo · reposo

**-58.3%**
tiempo · recorrido

**27**
repeticiones

| **Escenario** (mediana) | **Muestreo fijo** | **Adaptativo** |
|---|---|---|
| Reposo | 103.771 ms | 60.824 ms |
| Recorrido | 45.604 ms | 19.010 ms |

`terrain_sampling_proof`: `default.json`, gravedad $(0, 392)$, 10 subpasos, 8/3 iteraciones, 180 pasos de calentamiento y 240 medidos.

Es una medición de ese nivel y ese hardware, no una garantía universal.

# 03 DEPLOYMENT

El mismo código C++ corre nativo y en el navegador, con servicios en AWS para ranking y niveles.

# UNA BASE DE CÓDIGO, DOS DESTINOS

## NATIVO

C++23, CMake y raylib; `just` como atajo.

```
just build
just run       # build/native/ray
```

Pruebas numéricas como objetivos extra:

```
cmake --preset native-debug
./build/native/terrain_geometry_proof
./build/native/terrain_sampling_proof
```

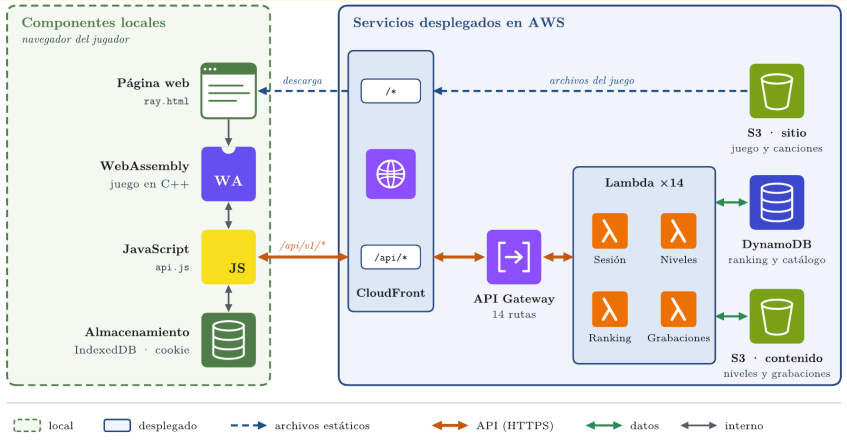
## WEB: WEBASSEMBLY

Emscripten compila el mismo código y precarga los assets.

```
source emsdk/emsdk_env.sh
just web
just serve     # :6931/ray.html
```

- Hace falta un servidor: con `file://` no se carga el filesystem virtual.
- JavaScript guarda la persistencia del navegador y habla con la API.
- `web-release`, `aws-deploy-api`, `aws-deploy-web`.

# ARQUITECTURA EN AWS




CloudFront sirve el juego desde S3 y enruta `/api/v1/*` a API Gateway → Lambdas (Node.js) → DynamoDB y S3.

# LA API DEL JUEGO

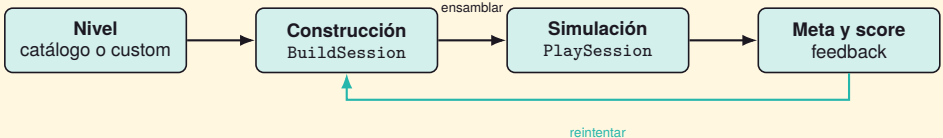
| **Ruta** (`/api/v1`) | **Resultado** |
|---|---|
| `GET /health` | estado del servicio |
| `POST /session` | identidad y cookie |
| `GET /levels/{id}` | definición del nivel |
| `POST /levels` | publica un nivel |
| `PUT …/attempts/{attemptId}` | valida y guarda un intento |
| `PUT …/recording` | sube una grabación |
| `GET …/leaderboard` | ranking del nivel |

- Lectura y escritura separadas: consultar un ranking no modifica un nivel.
- Escrituras exigen el `Origin` configurado y `x-combi-request: 1`.
- Estado de dominio en **DynamoDB**; grabaciones por hash en **S3**.
- El juego funciona en local sin desplegar nada: AWS es una ruta de entrega, no una dependencia del motor.

# 04 GAMEPLAY

Construir, simular, llegar a la meta y competir con lo que publica la comunidad.

# EL CICLO DE JUEGO













`PhysicsAssembler` traduce la grilla en cuerpos y joints; la interfaz nunca inventa resultados físicos.

# PIEZAS CON RESPONSABILIDAD

| Pieza | Papel en el sistema |
|---|---|
| Cabina del conductor | aloja al conductor; condición de meta |
| Bloque de pasajero | transporta un pasajero |
| Rueda | contacto circular + revolute joint |
| Ventilador / vela | activaciones de propulsión |
| TNT | explosión, impulsos, rotura en cadena |
| Conectores | compatibilidad y creación de joints |

- Una pieza no es una imagen: define masa, forma, conectores, orientación y activación.
- Construir: colocar, quitar, rotar, seleccionar y reposicionar sobre el `EditorGrid`.
- Las activaciones pasan por `ActivationRegistry`, separadas de la integración física.

# LLEGAR A LA META Y SUMAR PUNTOS

### GOALWINDOWTRACKER

- Compara la muestra del frame anterior y la actual: detecta entradas *entre* frames.
- Al entrar el conductor (o su vehículo) arranca una ventana de **3 s** que no se cancela si sale.
- Piezas y pasajeros que tocan la meta cuentan como entregados.

### SCORERULES.CPP

500 por pasajero recogido · 1000 por entregado · hasta 3000 por piezas entregadas · hasta 3000 por rapidez ($t < 30$ s)

$$S_{earned} = S_{pickup} + S_{delivery} + S_{parts} + S_{speed}$$

$$S_{total} = \mathrm{round}\left( \frac{S_{earned}\, M}{M + \max(0,\ N_{initial} - 2)} \right),\ M = 25$$

Una combi enorme se penaliza. Las reglas se comparten con el backend mediante vectores de prueba.

# EDITOR DE NIVELES, RANKING Y FANTASMA

- El editor define terreno, zona de construcción, meta, pasajeros y música.
- Los niveles se guardan en JSON y se validan antes de publicarse.
- Partidas campeonas grabadas a **20 Hz**; el fantasma reproduce la trayectoria, no re-simula.
- Ranking por nivel y niveles de la comunidad desde la API.






# CONCLUSIONES Y LÍMITES

## LO QUE LOGRAMOS

- Un juego recreativo como problema integrador de computación gráfica.
- Grilla discreta → cuerpos continuos; curva → malla visible → obstáculo físico, sin divergencias.
- Motor propio con todo el proceso expuesto: broad/narrow phase, manifolds, impulsos, joints y rotura local.
- Nativo y web desde el mismo código.

## LO QUE FALTA

- Solver iterativo: estable a la escala del juego, no exacto en todos los regímenes.
- Broad phase, iteraciones, discretización y umbrales de rotura afectan el resultado.
- Ampliar pruebas de regresión de escenas completas e instrumentación del motor.

# EQUIPO

**Alejandro Joel Ore Garcia**
Motor de físicas

**Kiara Alexandra Balcazar Santa Cruz**
Geometría computacional

**Juan Diego Castro Padilla**
Deployment

**Kayla De Vivanco Briceño**
Gameplay

**Docente** Itamar Franco Salazar Reque
**Coordinador** Eric Javier Biagioli

# ¡GRACIAS!