Simulación de Monte-Carlo en el Grid

Aproximación del número Pi y más

Autor: **CÉSAR GONZÁLEZ SEGURA**
<cegonse@posgrado.upv.es>
Conceptos de la **C**omputación en el **G**rid y **C**loud
MU en Computación Paralela y Distribuida

ÍNDICE

MARCO TEÓRICO

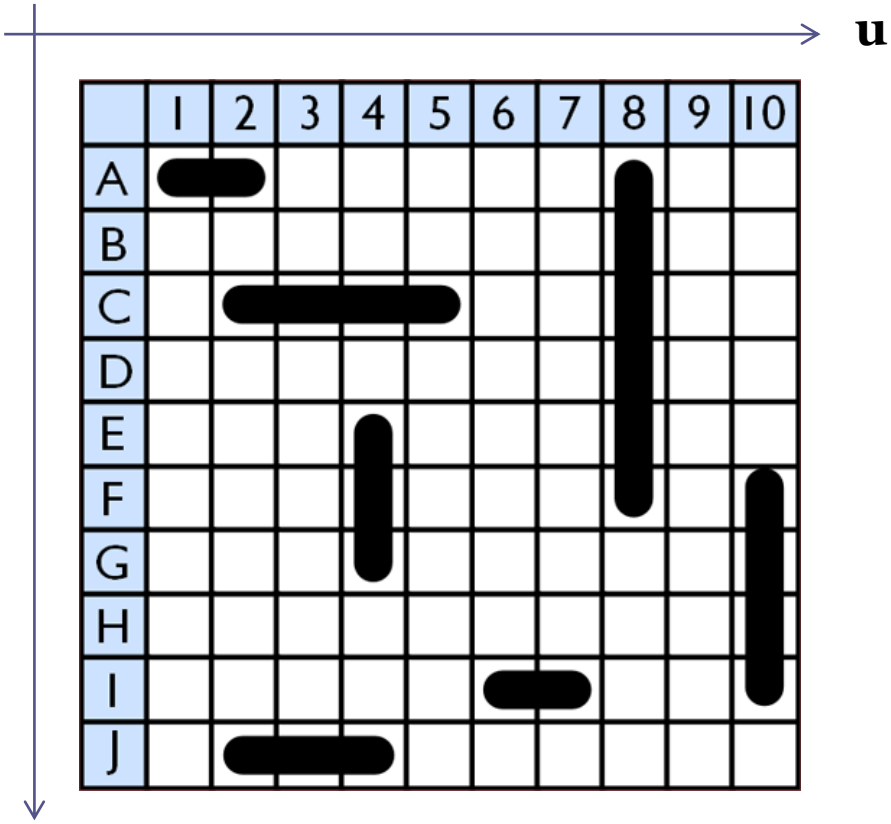
Método de Monte-Carlo

"Método probabilístico para evaluar expresiones matemáticas complejas."

- Se evalúa el valor de la función en puntos aleatorios siguiendo cierta distribución de probabilidad.
- Cuanto mayor sea el número de puntos, más se asemejará la solución aproximada a la solución real.

MARCO TEÓRICO

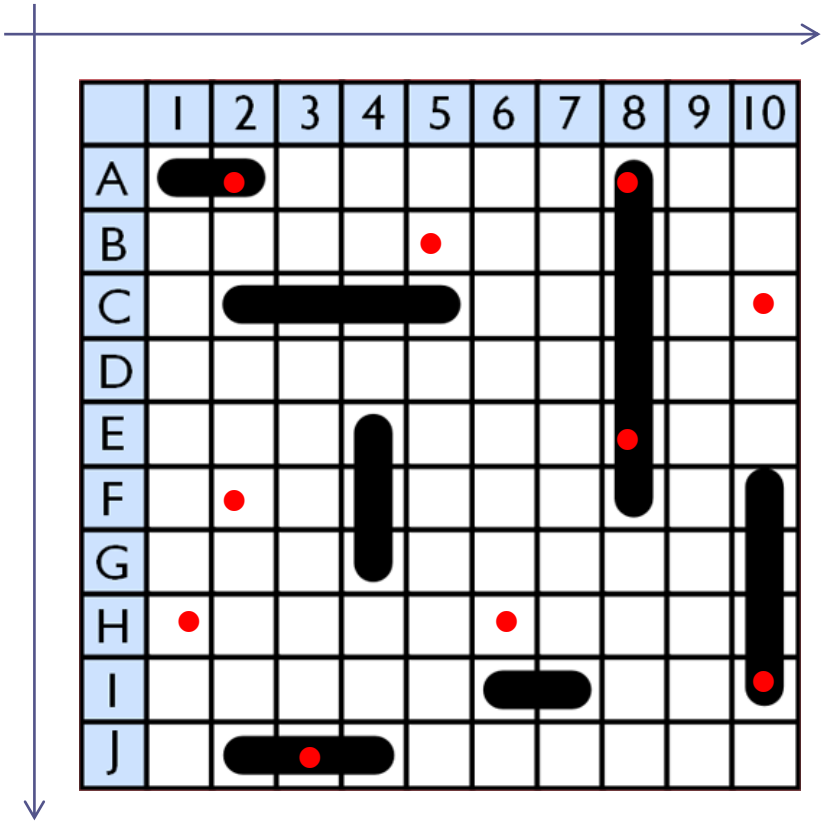
Ejemplo: Hundir la Flota



- Obtenemos N*N puntos aleatorios distribuidos en el rango 1~10 y A~J respectivamente.
- Evaluamos el resultado obtenido (agua o tocado).

MARCO TEÓRICO

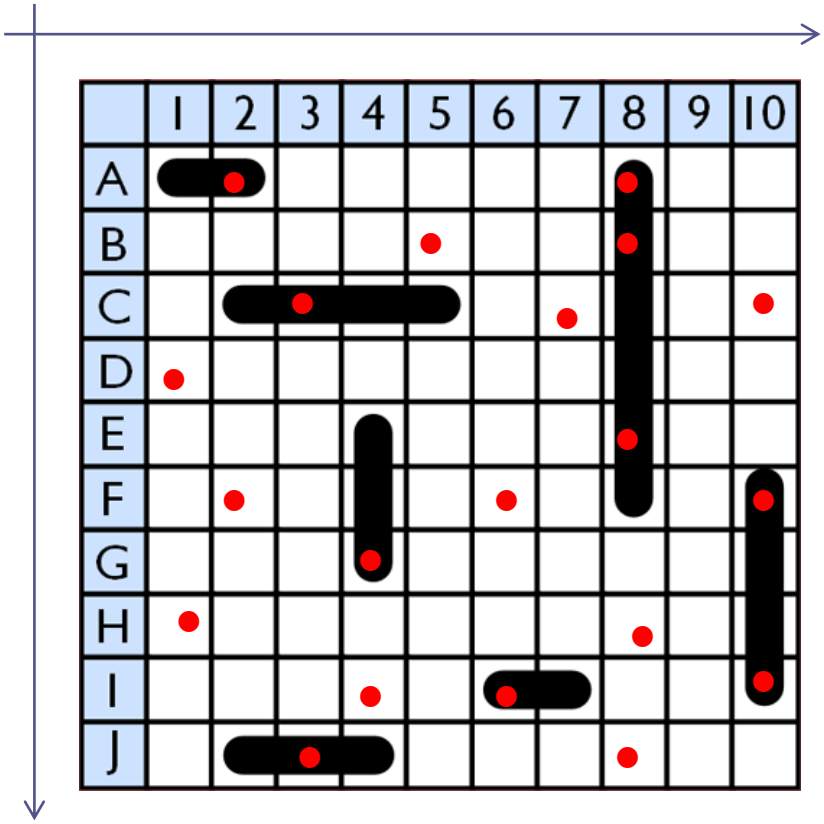
Ejemplo: Hundir la Flota



- N = 10.
- Empieza a distinguirse la topología del mapa (posiciones de barcos, zonas de agua…).
- Se ha obtenido una solución inicial pero con mucho error.

MARCO TEÓRICO

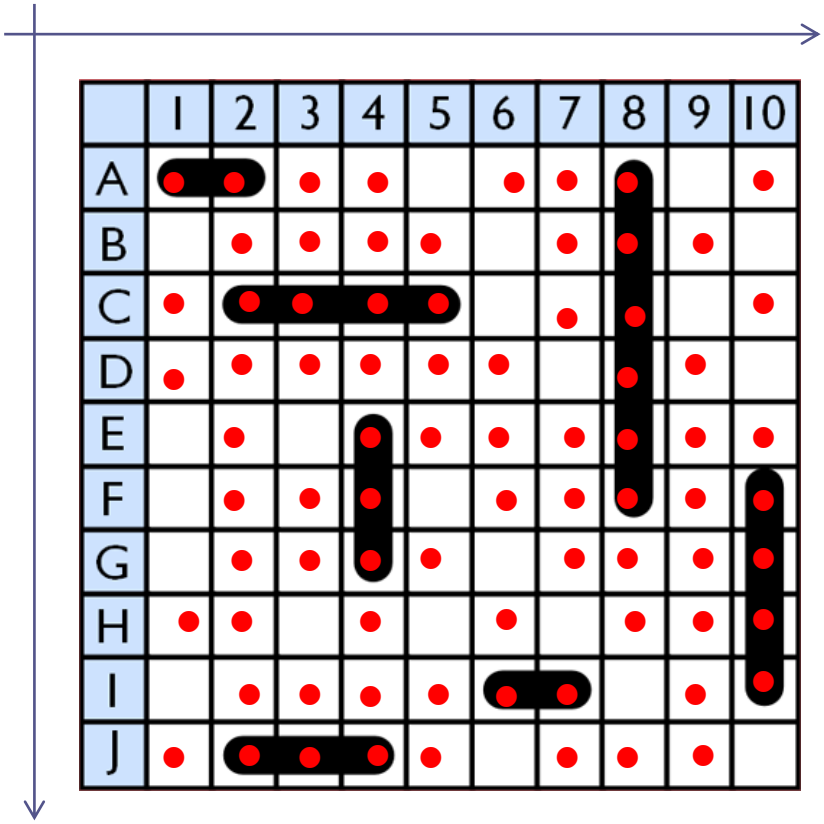
Ejemplo: Hundir la Flota



- N = 20.
- Con un número mayor de puntos se conoce mucha más información.
- Se ha disminuido el error en la solución obtenida.

MARCO TEÓRICO

Ejemplo: Hundir la Flota



- N grande.
- Con un número de puntos los suficientemente grande, se conoce casi por completo la topología del mapa.
- Se ha aproximado la solución con mucha exactitud.

OBJETIVOS Y SOLUCIÓN PROPUESTA

Objetivo Principal

"Gridificar" el método de Monte-Carlo para acelerar su ejecucción.

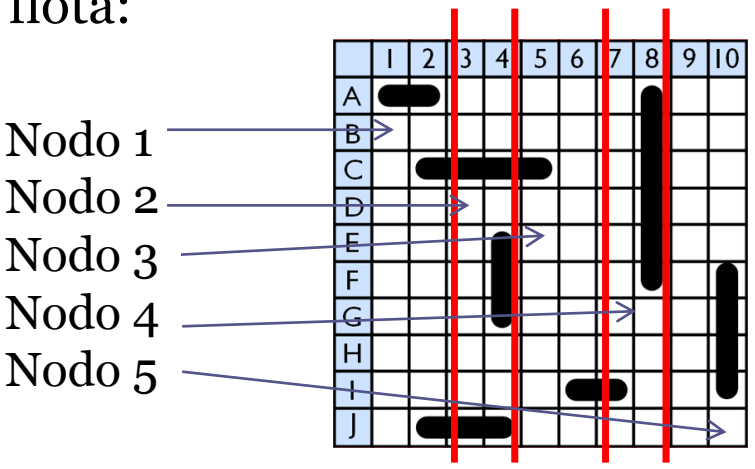
- El problema se puede dividir en sub-problemas de grano grueso: es un problema *poco acoplado*.
- Al tratar con números aleatorios pueden surgir problema (semillas de generadores de números aleatorios entre otros).

OBJETIVOS Y SOLUCIÓN PROPUESTA

Solución Propuesta

División del problema mediante *divide y vencerás*.

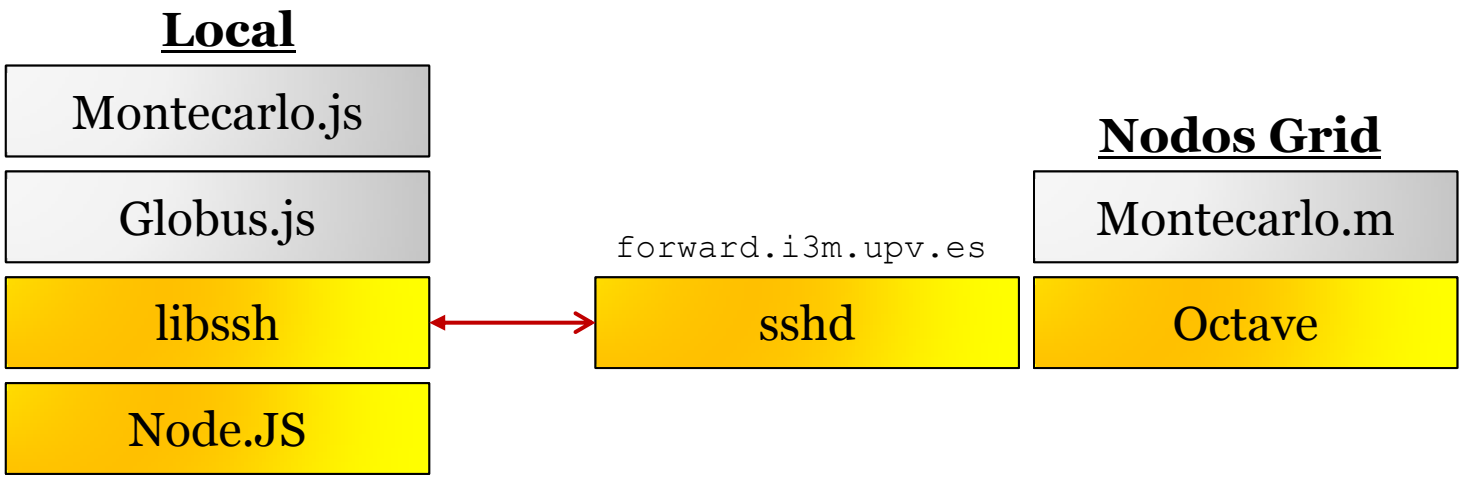
- La distribución aleatoria se ha dividido en *N* distribuciones acotadas en los rangos apropiados para el nodo.
- Véase el ejemplo de hundir la flota:



DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

Diseño del Sistema

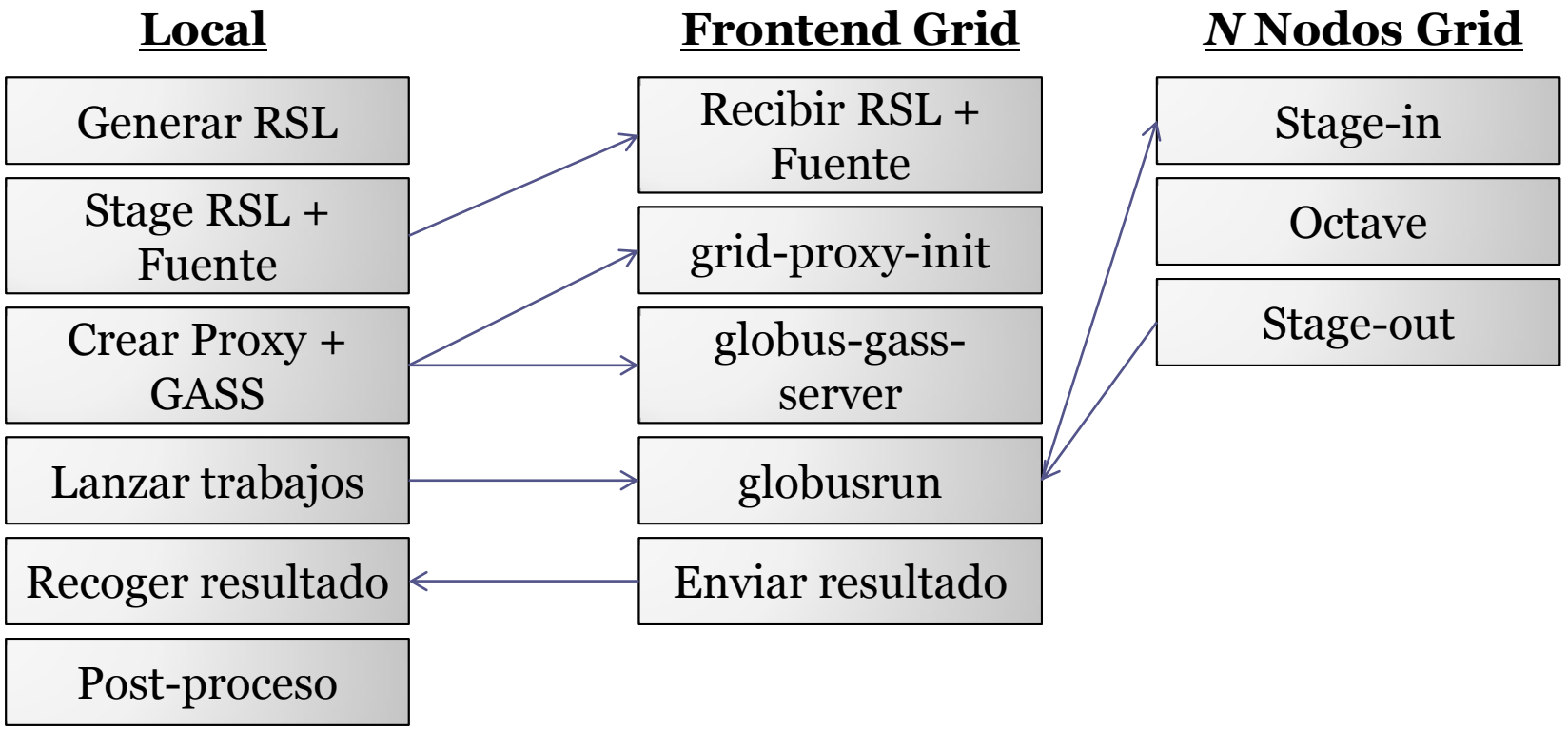
Frontend en local + runtime en el Grid



- Primera versión en C++.
- Problemas al tener que compilar: ¿C+11? ¿GSL? ¿librandom? ¿libssh? ¿libssh2? ¿boost? ¿mkl? …
- Tras varios problemas en la comunicación: paso a Node.js.

DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

Diagrama de flujo del sistema



DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

Detalles de Implementación

- Código fuente en Gitlab del DSIC:
  https://gitlab.dsic.upv.es/cegonse/grid-montecarlo/tree/master

- Librerías utilizadas:
  - SSHv2: Acceso SSH2 desde Node.JS.
  - Async: Gestión avanzada de tareas en Node.JS.
  - Readline-sync: Acceso a *stdin* mediante streams síncronos.

- Probado en Node.JS 4 bajo Linux.

DEMO: APROXIMACIÓN DEL NÚMERO PI

Método para aproximar

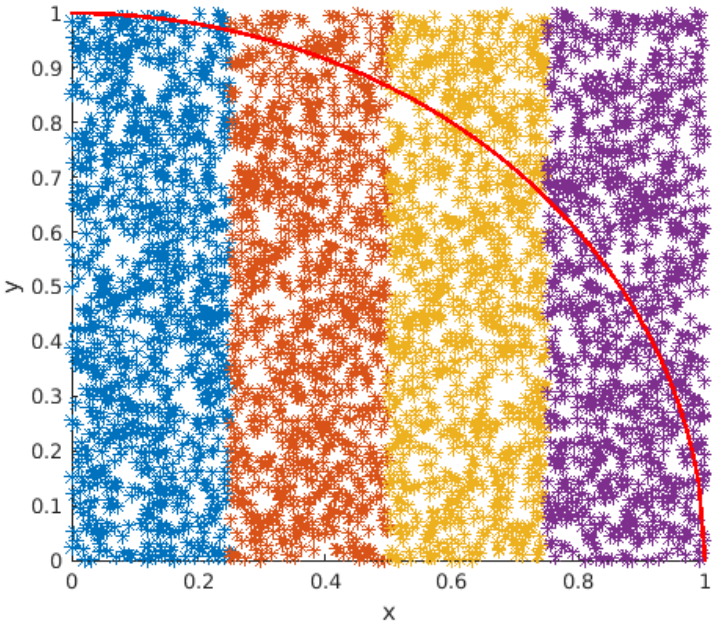
- Integración de la función del círculo para hallar su área (Pi).
- Los valores en los que evaluar la función son aleatorios (Montecarlo).

```
scale = 1 / nodes;

for i = 1:n
    x = scale*index + scale*rand();
    y = rand();

    a = sqrt(x*x + y*y);

    if a <= 1
        m_pi = m_pi + 1;
    end
end
```



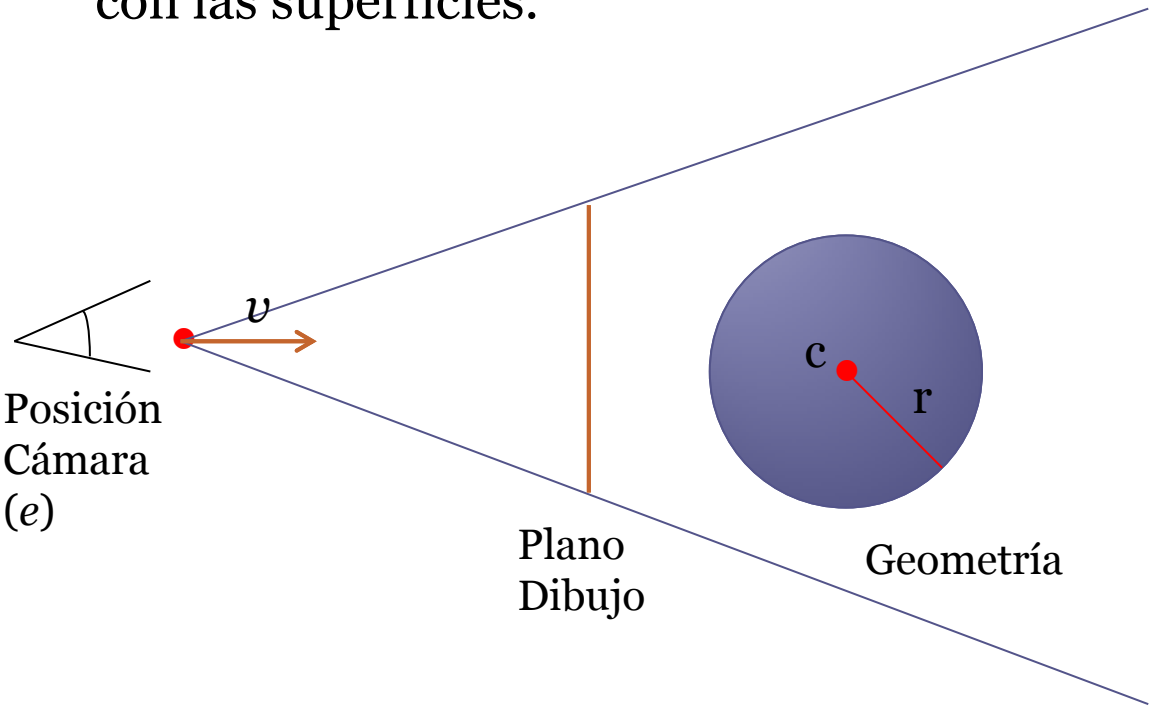
DEMO: APROXIMACIÓN DEL NÚMERO PI

Ejecucción de la demo
(Cruzemos los dedos)

DEMO: RENDERIZADO MEDIANTE PATH-TRACING

Qué es el trazado de rayos

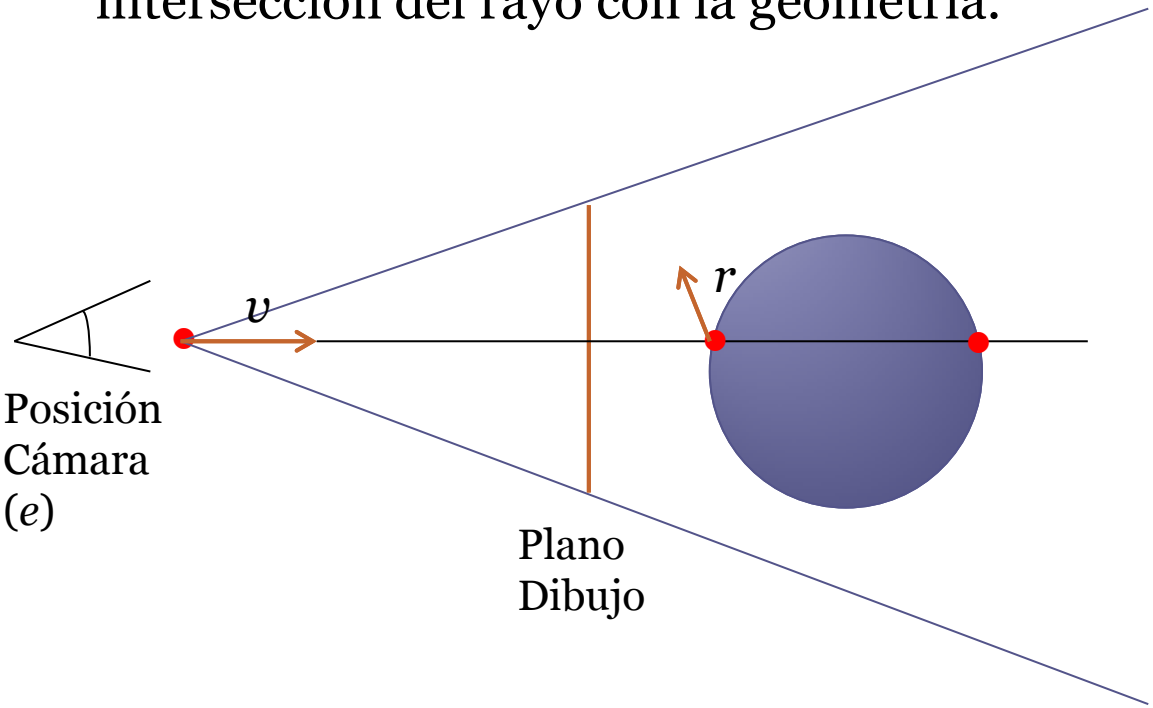
- Dibujado de escenas 3D simulando la interacción de los fotones con las superficies.



DEMO: RENDERIZADO MEDIANTE PATH-TRACING

Qué es el trazado de rayos

- Se lanzan rayos desde la cámara hacia la escena y se encuentra la intersección del rayo con la geometría.



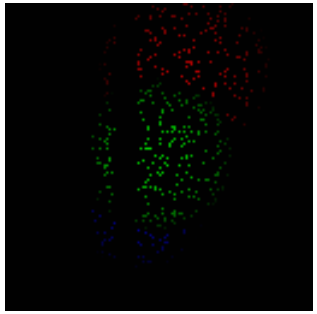
- r : vector de reflexión
Cantidad de luz que refleja la superficie:
  - *Difusa*
  - *Especular*
  - *Reflejos*

DEMO: RENDERIZADO MEDIANTE PATH-TRACING

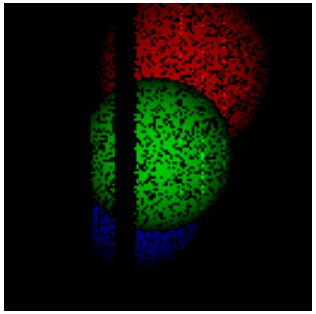
Qué es el path-tracing

- Aplicación del método de Monte-Carlo al trazado de rayos.
- Se generan rayos de manera aleatoria, asegurando que intersecan con el plano de dibujo. Cuantas más muestras aleatorias se generen, más se asemejará la imagen al resultado final.
- En la demo: escena con tres esferas, material difuso (sin reflejos).

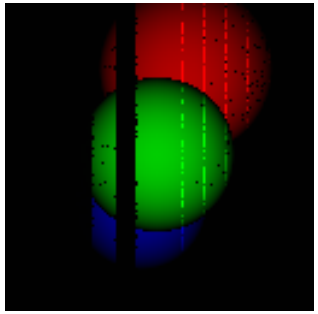
$N = 100$



$N = 1000$



$N = 2500$



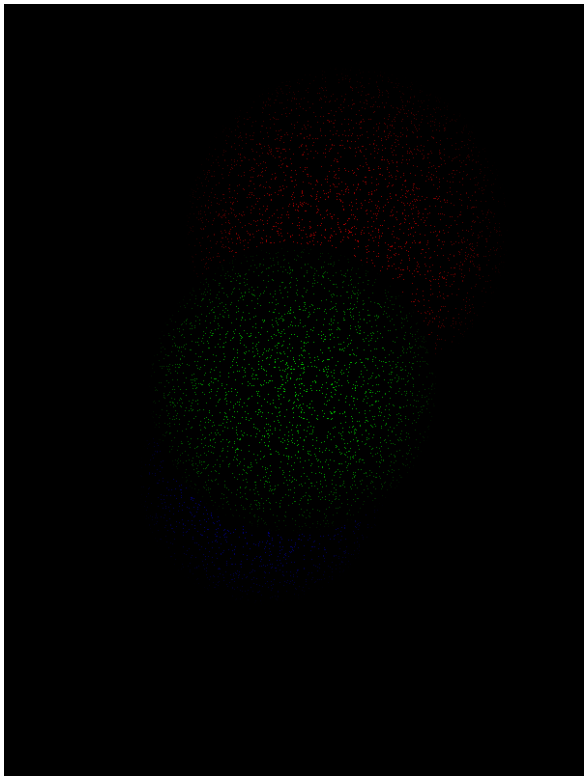
DEMO: RENDERIZADO MEDIANTE PATH-TRACING

Ejecucción de la demo
(Cruzemos los dedos)

RESULTADOS

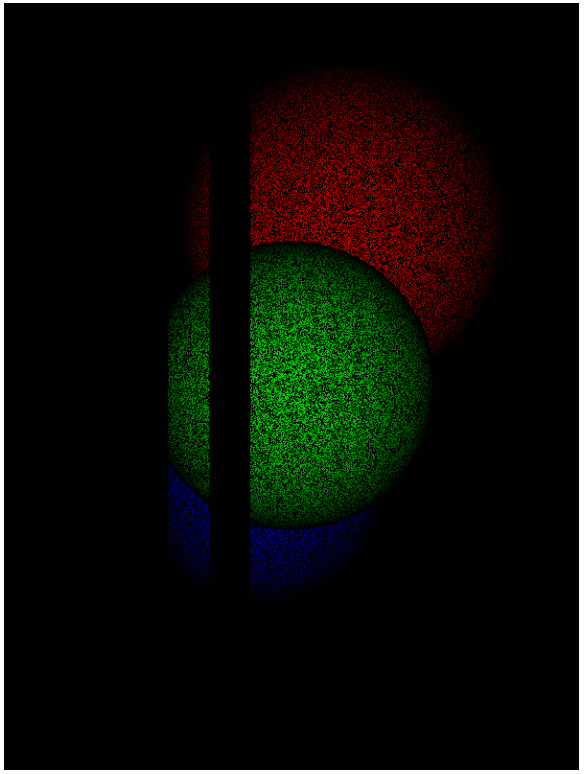
Comparación ejecucción local – Grid

Local



$N = 20000$

Grid



CONCLUSIONES

Validez del método

- El método de Monte-Carlo ha demostrado ser un buen problema para **Gridificar**.
- Usando un tiempo de ejecucción similar se pueden obtener mejores resultados obteniendo los cálculos en el Grid.
- Es importante tener cuidado con la *no-aleatoriedad* de los generadores de números aleatorios y con la disponibilidad de los nodos de cómputo.

¡Muchas gracias por vuestra atención!

¿Preguntas?