Formule del Raycasting - Guida Completa

1. Coordinate e Sistema di Riferimento

Prima di tutto, stabiliamo il nostro sistema di coordinate:

- **Player Position**: (px, py) - posizione del giocatore nella mappa 2D
- **Player Angle**: θ (theta) - direzione in cui guarda il giocatore, in radianti
- **Map**: griglia bidimensionale dove 0 = spazio vuoto, 1 = muro

2. Calcolo dell'Angolo del Raggio

Per ogni colonna i dello schermo (da 0 a SCREEN_WIDTH-1):

```
ray_angle = player_angle + FOV_half - (i * FOV / SCREEN_WIDTH)
```

Spiegazione:

- FOV = Field of View (campo visivo, tipicamente 60°)
- FOV_half = FOV / 2
- Partiamo dall'estrema sinistra del campo visivo e ci spostiamo gradualmente verso destra

Esempio pratico:

- Se FOV = 60° = π/3 radianti
- Per schermo di 320 pixel
- Colonna 0: angolo più a sinistra
- Colonna 160: angolo centrale (direzione del giocatore)
- Colonna 319: angolo più a destra

3. Direzione del Raggio

Una volta calcolato l'angolo, troviamo la direzione del raggio:

```
ray_dx = cos(ray_angle)
ray_dy = sin(ray_angle)
```

Significato:

- ray_dx e ray_dy formano un vettore unitario (lunghezza 1) che indica la direzione

- cos(θ) ci dà la componente orizzontale del movimento
- sin(θ) ci dà la componente verticale del movimento

4. Algoritmo DDA - Le Formule Chiave

4.1 Inizializzazione delle Variabili

```
// Posizione attuale del raggio
current_x = px
current_y = py

// Quale cella della griglia stiamo controllando
map_x = floor(current_x)
map_y = floor(current_y)

// Calcolo delle distanze delta
delta_dist_x = |1 / ray_dx|
delta_dist_y = |1 / ray_dy|
```

Cosa rappresenta delta_dist:

- delta_dist_x = distanza che il raggio percorre per attraversare una cella orizzontalmente
- delta_dist_y = distanza che il raggio percorre per attraversare una cella verticalmente

4.2 Calcolo delle Distanze Iniziali e Direzioni di Step

Per la direzione X:

```
if ray_dx < 0:
    step_x = -1
    side_dist_x = (current_x - map_x) * delta_dist_x
else:
    step_x = 1
    side_dist_x = (map_x + 1.0 - current_x) * delta_dist_x
```

Per la direzione Y:

```
if ray_dy < 0:
    step_y = -1
    side_dist_y = (current_y - map_y) * delta_dist_y
else:
    step_y = 1
    side_dist_y = (map_y + 1.0 - current_y) * delta_dist_y
```

Cosa rappresentano:

- step_x, step_y: in quale direzione ci muoviamo nella griglia (+1 o -1)
- side_dist_x, side_dist_y: distanza per raggiungere il prossimo bordo di cella in quella direzione

4.3 Il Loop Principale DDA

```
while (hit == 0):
    // Quale direzione è più vicina?
    if side_dist_x < side_dist_y:
        side_dist_x += delta_dist_x
        map_x += step_x
        side = 0  // abbiamo colpito un lato verticale
    else:
        side_dist_y += delta_dist_y
        map_y += step_y
        side = 1  // abbiamo colpito un lato orizzontale

    // Controlla se abbiamo colpito un muro
    if map[map_x][map_y] > 0:
        hit = 1
```

5. Calcolo della Distanza Finale

Una volta trovato il muro, calcoliamo la distanza percorsa:

```
if side == 0:
    perp_wall_dist = (map_x - px + (1 - step_x) / 2) / ray_dx
else:
    perp_wall_dist = (map_y - py + (1 - step_y) / 2) / ray_dy
```

Attenzione: Usiamo la distanza perpendicolare, non la distanza euclidea, per evitare l'effetto "fish-eye" (distorsione ai bordi dello schermo).

6. Calcolo dell'Altezza della Parete

```
line_height = SCREEN_HEIGHT / perp_wall_dist
```

Posizionamento verticale:

```
draw_start = (-line_height / 2) + (SCREEN_HEIGHT / 2)
draw_end = (line_height / 2) + (SCREEN_HEIGHT / 2)

// Clamp ai bordi dello schermo
if draw_start < 0: draw_start = 0
if draw_end >= SCREEN_HEIGHT: draw_end = SCREEN_HEIGHT - 1
```

7. Formule per il Texturing (Avanzato)

Se vuoi aggiungere texture alle pareti:

7.1 Calcolo del Punto di Impatto

```
if side == 0:
    wall_x = py + perp_wall_dist * ray_dy
else:
    wall_x = px + perp_wall_dist * ray_dx

wall_x = wall_x - floor(wall_x)  // parte frazionaria
```

7.2 Coordinata della Texture

```
tex_x = wall_x * TEX_WIDTH
if (side == 0 && ray_dx > 0) || (side == 1 && ray_dy < 0):
    tex_x = TEX_WIDTH - tex_x - 1
```

8. Correzione Fish-Eye - Formula Chiave

Per evitare la distorsione fish-eye, moltiplichiamo la distanza per il coseno dell'angolo:

```
corrected_distance = raw_distance * cos(ray_angle - player_angle)
```

Perché funziona: I raggi ai bordi del campo visivo sono più lunghi di quelli centrali quando proiettati su uno schermo piatto. La correzione coseno compensa questa differenza.

9. Conversione Angoli

Ricorda le conversioni tra gradi e radianti:

```
radianti = gradi * π / 180
gradi = radianti * 180 / π
```

Angoli tipici:

- 0° = 0 rad = Est
- 90° = π/2 rad = Nord
- 180° = π rad = Ovest
- 270° = 3π/2 rad = Sud

10. Ottimizzazioni Matematiche

10.1 Precalcolo di Seni e Coseni

```
// All'inizializzazione del gioco
for i in range(0, 360):
    sin_table[i] = sin(i * π / 180)
    cos_table[i] = cos(i * π / 180)
```

10.2 Divisione Rapida per Altezza

Invece di dividere, moltiplica per l'inverso precalcolato:

```
inv_dist = 1.0 / perp_wall_dist
line_height = SCREEN_HEIGHT * inv_dist
```

Queste formule costituiscono il cuore matematico del raycasting. La bellezza sta nel fatto che trasformano un problema 3D complesso in calcoli 2D relativamente semplici!