DCC192

2025/1

UFMG

Desenvolvimento de Jogos Digitais

A6: Movimentação de Objectos Rígidos

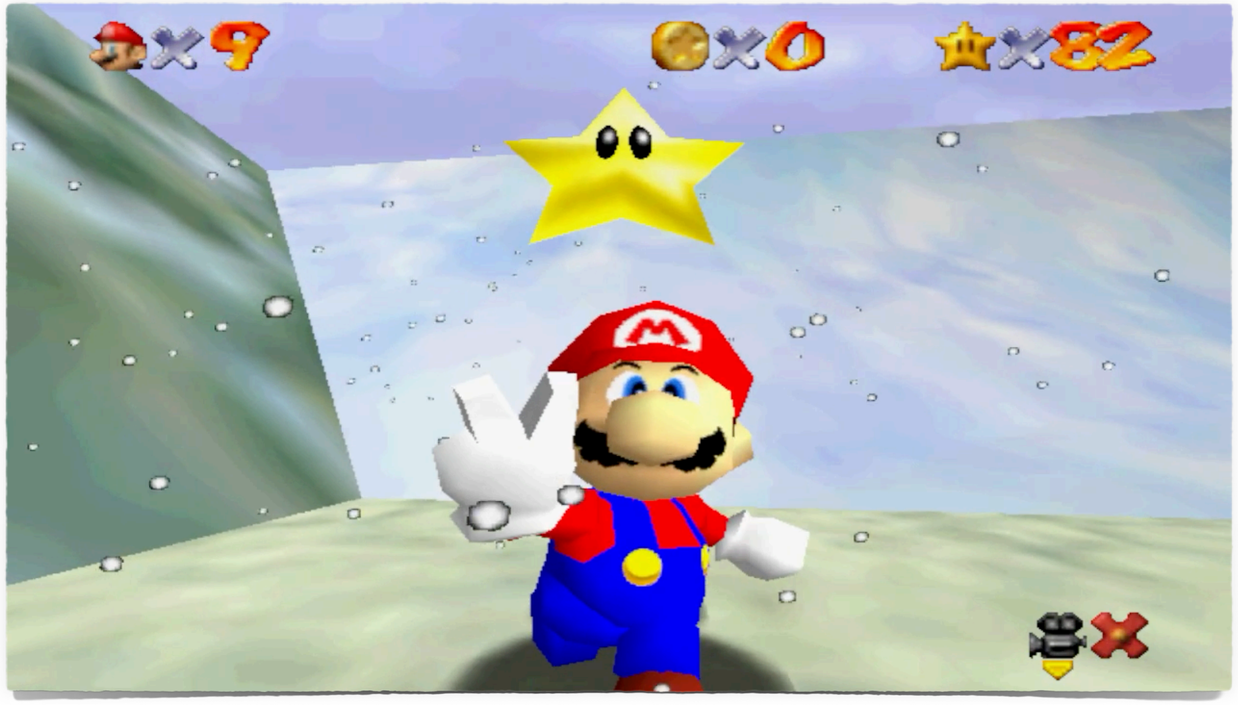
Prof. Lucas N. Ferreira

Plano de aula

- Física em Jogos Digitais
- Objetos Rígidos
  - Movimentação
  - Método de Euler Semi-Implícito
  - Aceleração da gravidade
  - Atrito
  - Resistência do Meio

Física em Jogos Digitais

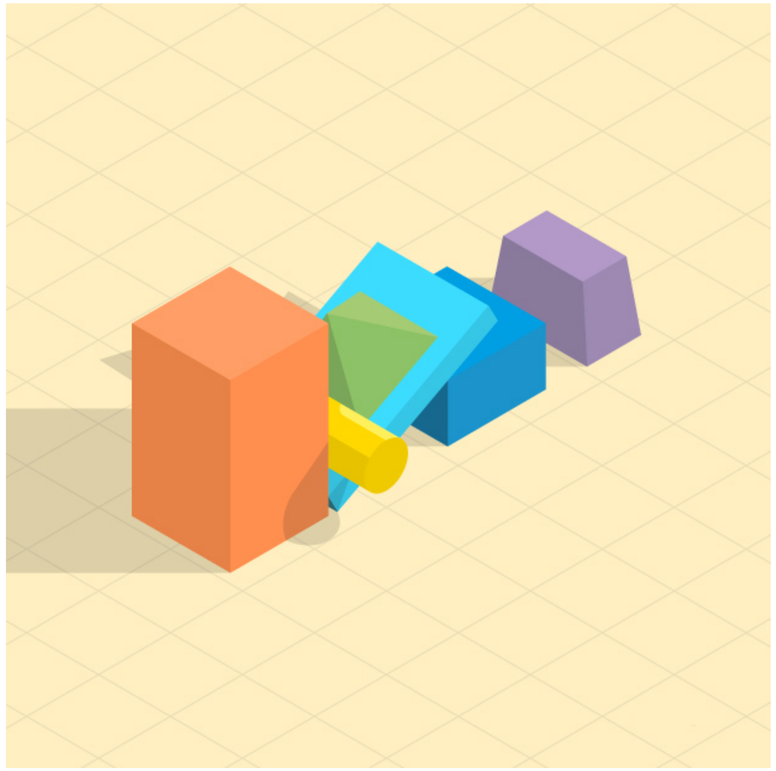
Geralmente, em jogos digitais queremos **mover objetos rígidos** por meio de aplicação de **forças** causadas pelo jogador ou por outros objetos do jogo:



- Andar/Correr
- Pular
- Deslizar
- Planar
- Vento
- ...

Objetos Rígidos

Objetos rígidos são solidos que não sofrem deformação.



As propriedades de objetos rígidos são:

- **Massa** (escalar): quantidade de matéria no corpo
- **Posição** (vetor): localização no espaço (2D ou 3D)
- **Velocidade** (vetor): taxa de variação de posição
- **Aceleração** (vetor): taxa de variação de velocidade

Apesar de objetos rígidos não existirem na vida real, são excelentes simplificações para simulações de objetos em jogos.

Movimentação de objetos rígidos

A movimentação de objetos rígidos pode ser descrita pela Física Newtoniana:

Primeira Lei de Newton:

Um objeto em repouso permanece em repouso, ou se estiver em movimento, permanece em movimento com velocidade constante, a menos que uma força externa atue sobre ele.

Segunda Lei de Newton:

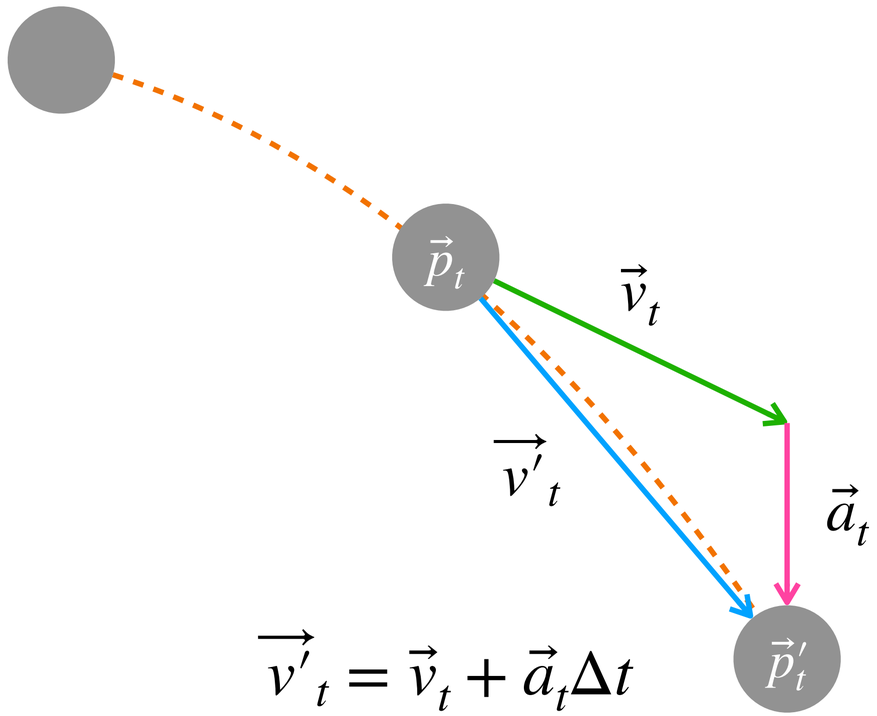
Força ($\vec{f}$) é igual a massa m vezes a aceleração $\vec{a}$: $\vec{f} = m\vec{a}$

```
RigidBody::Update(float dt) {
  mVelocity += mAccelaration * dt;
  mPosition += mVelocity * dt;
  mAcceleration.Set(0f, 0f);
}


RigidBody::ApplyForce(Vector2 f) {
  mAccelaration += f * 1f/mMass;
}
```

Formalmente, estamos integrando numericamente uma equação diferencial ordinária de movimento usando o **Método de Euler Semi-Implícito**

Método de Euler Semi-implícito



$$\overrightarrow{v'}_t = \vec{v}_t + \vec{a}_t \Delta t$$

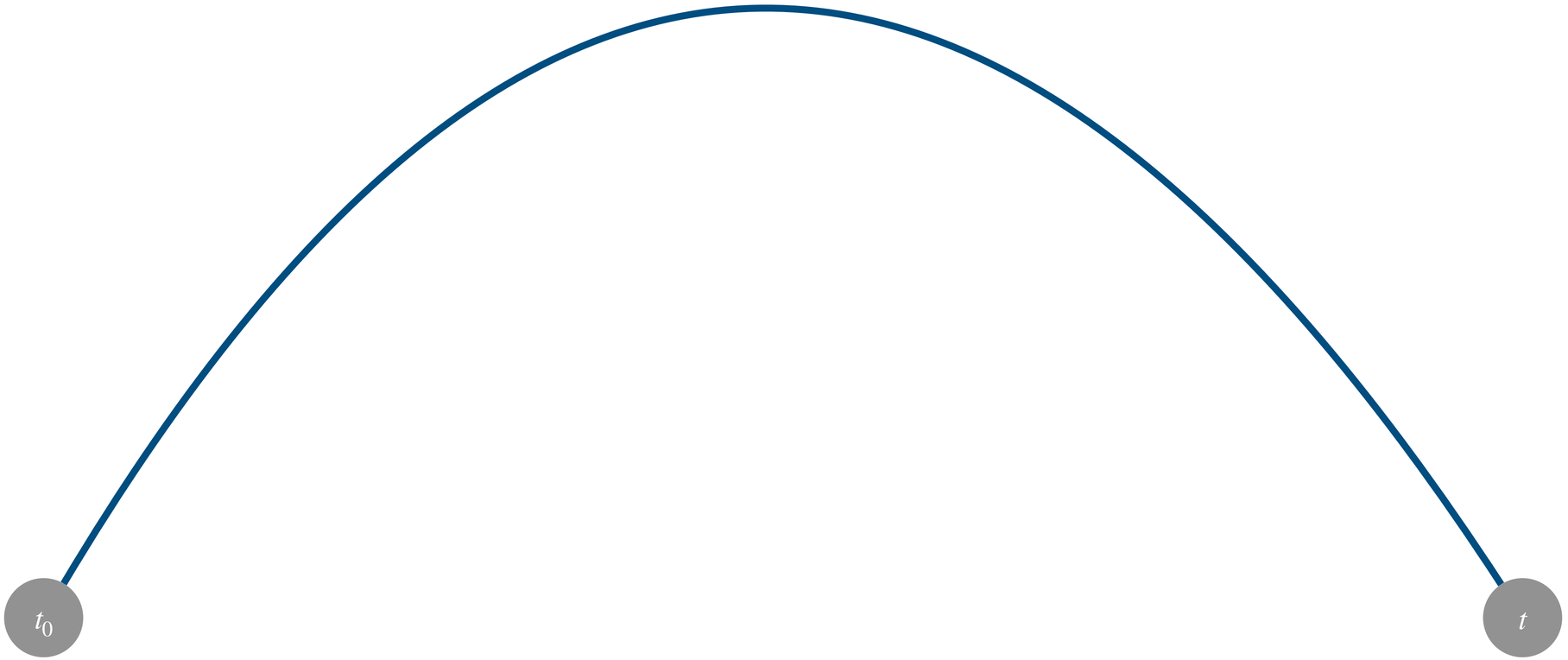
$$\overrightarrow{p'}_t = \vec{p}_t + \overrightarrow{v'}_t \Delta t$$

```
// Euler Semi-implícito
RigidBody::Update(float dt) {
  mVelocity += mAccelaration * dt;
  mPosition += position * dt;
  mAcceleration.Set(0f, 0f);
}
```

- Assumimos que a velocidade e a aceleração são constantes entre os quadros;
- Os movimentos são divididos em uma sequência de retas;
- Quanto menor o Δt, melhor a aproximação do **movimento real**.

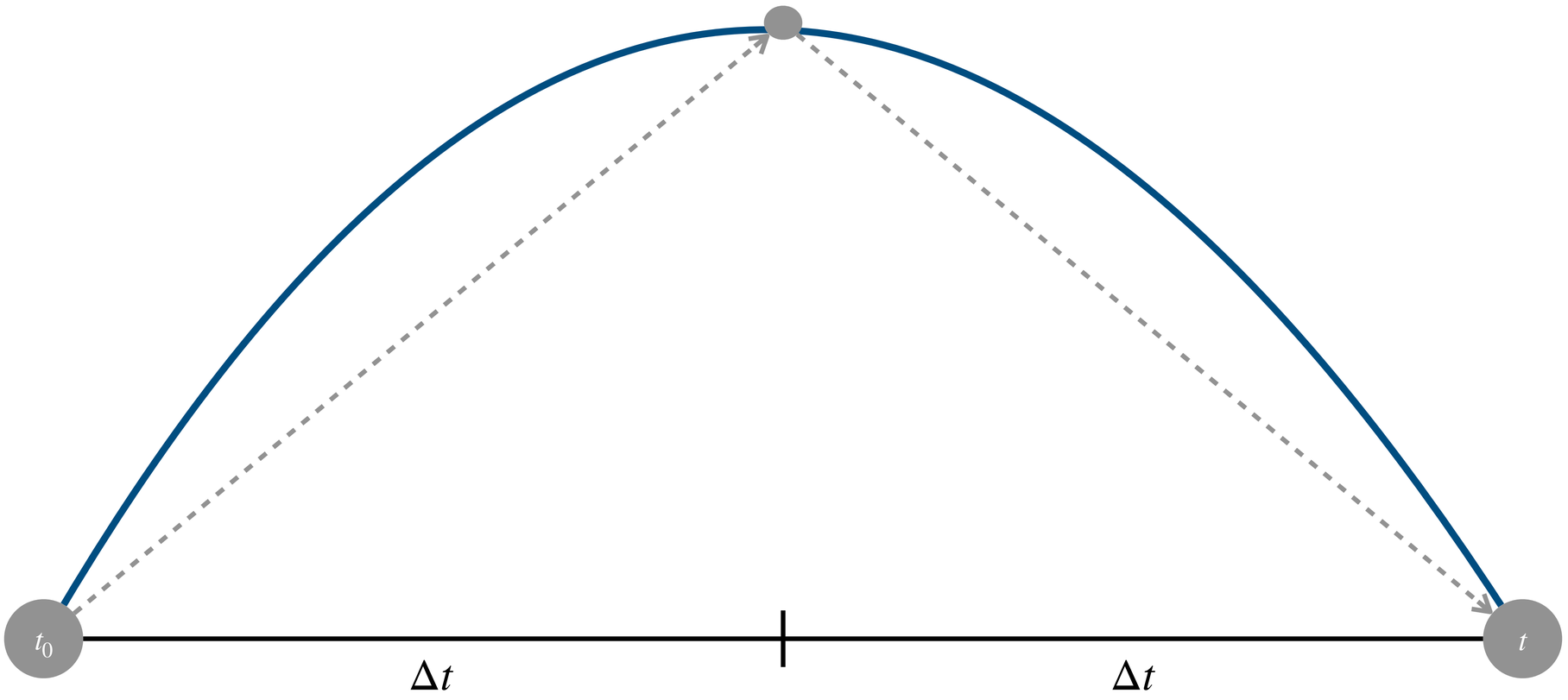
Impacto do tamanho do delta time

Quanto menor o Δt, melhor a aproximação do **movimento real**, ou seja, mais suave será o movimento.



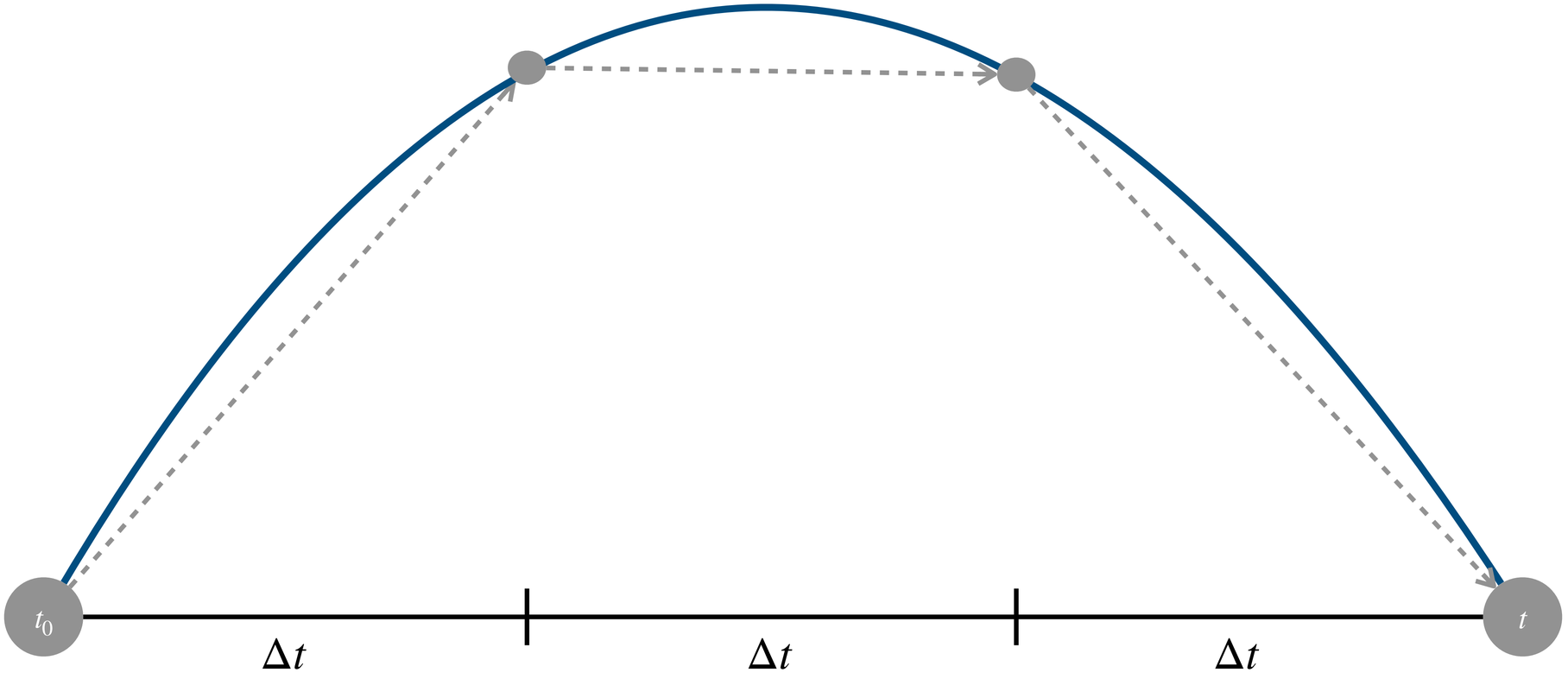
Impacto do tamanho do delta time

Quanto menor o Δt, melhor a aproximação do **movimento real**, ou seja, mais suave será o movimento.



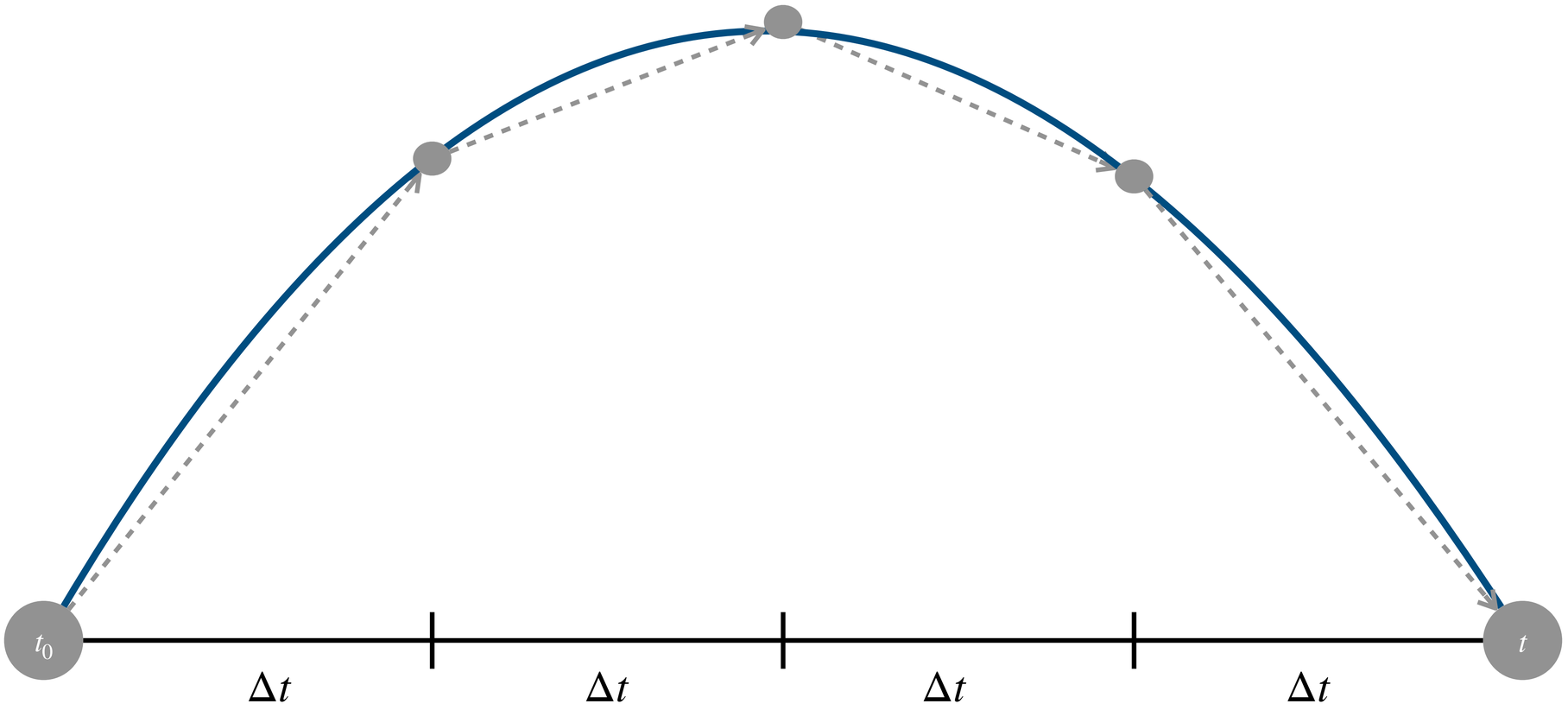
Impacto do tamanho do delta time

Quanto menor o Δt, melhor a aproximação do **movimento real**, ou seja, mais suave será o movimento.



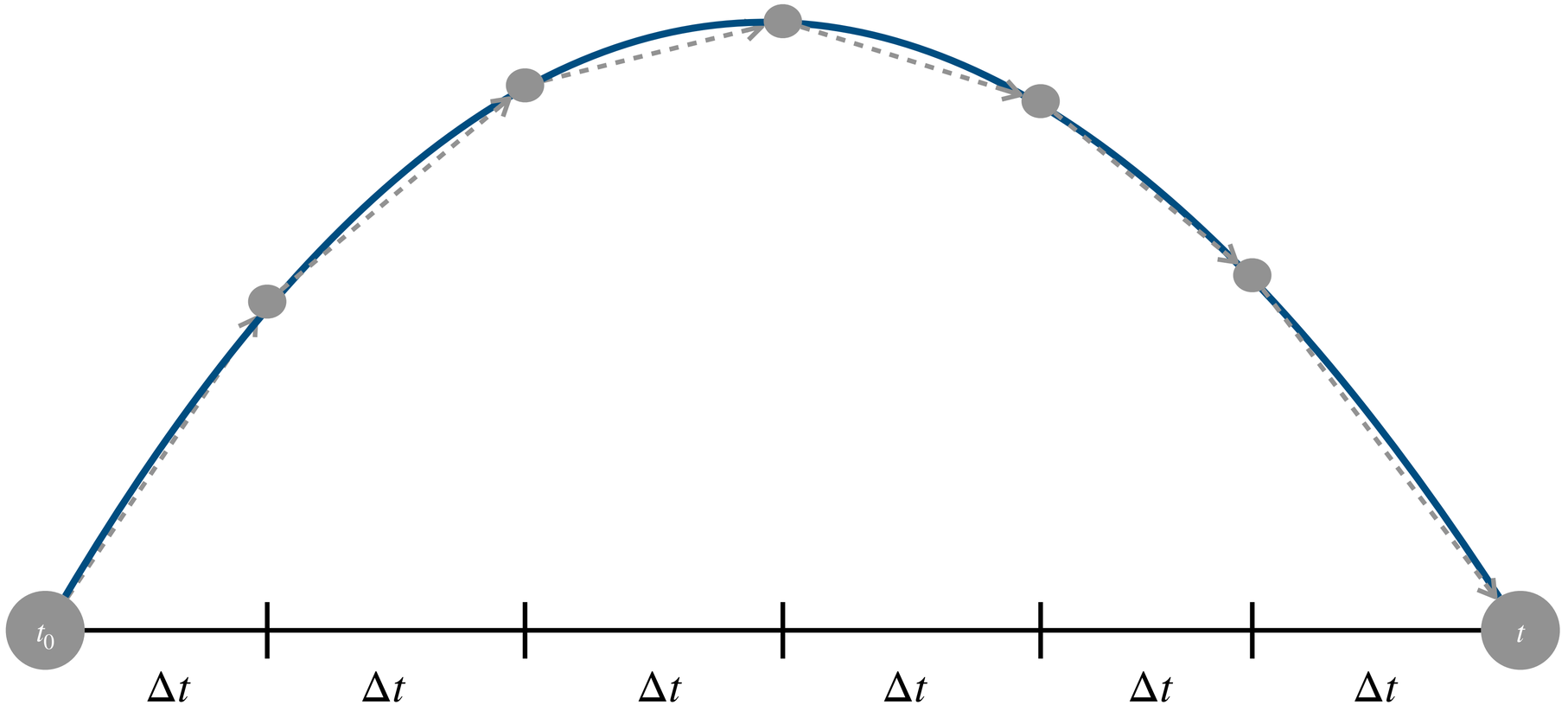
Impacto do tamanho do delta time

Quanto menor o Δt, melhor a aproximação do **movimento real**, ou seja, mais suave será o movimento.



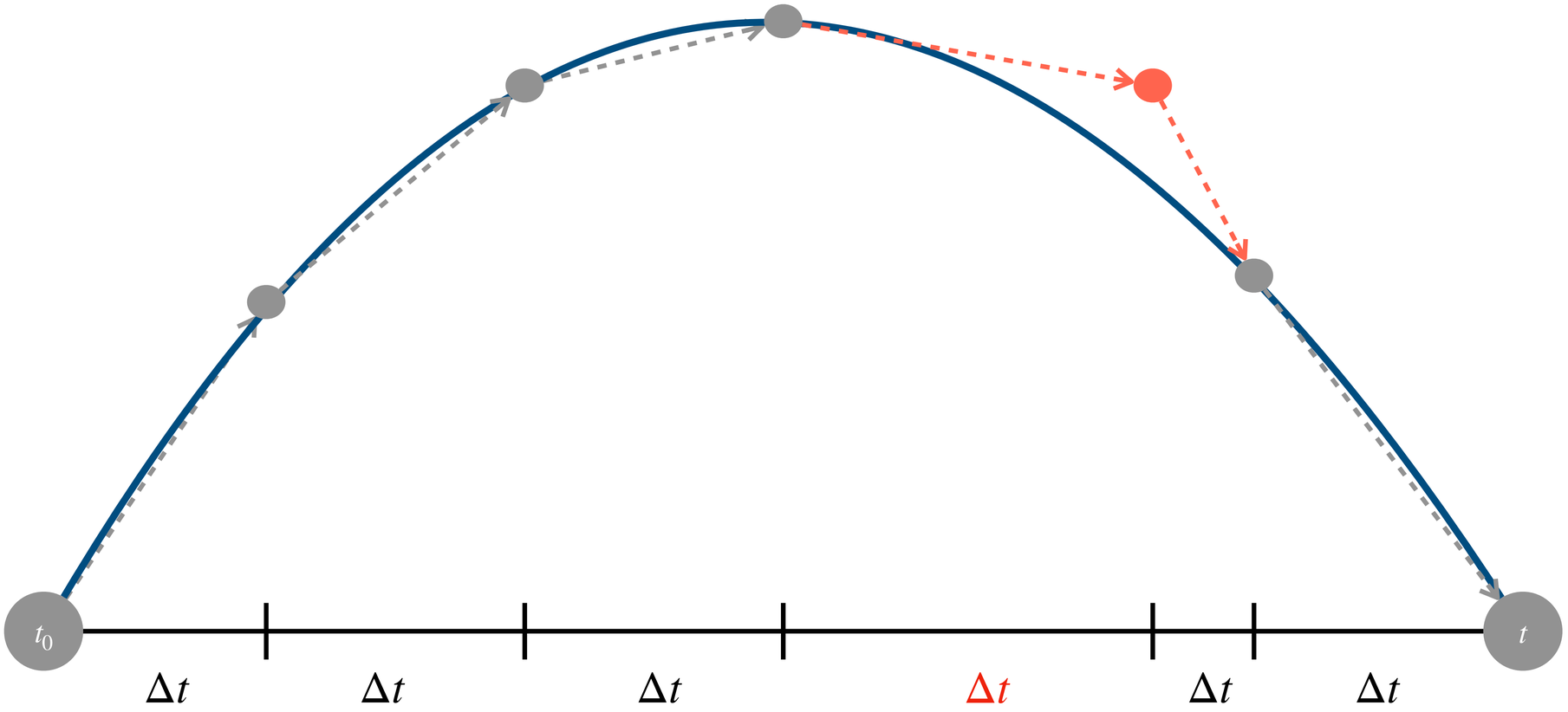
Impacto do tamanho do delta time

Quanto menor o Δt, melhor a aproximação do **movimento real**, ou seja, mais suave será o movimento.



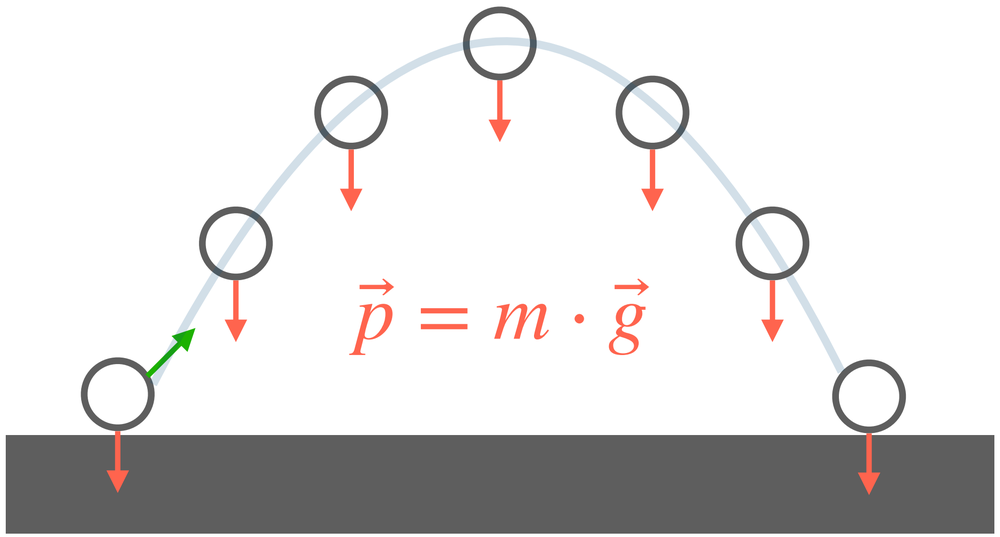
Impacto na variação do delta time

Como mencionados em aulas anteriores, se o delta time Δt for variável entre quadros, a simulação física pode ficar instável!



Aceleração da Gravidade

É muito comum jogos aplicarem uma **força peso** ao seus objetos, que é causada pela aceleração da gravidade.



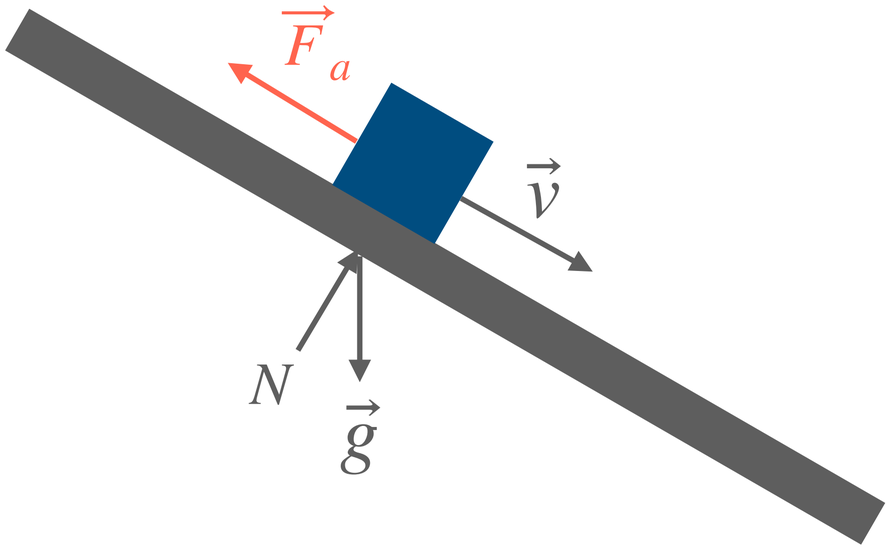

```
Vector2 g = Vector2(0.f, 9.8f);

RigidBody::Update(float dt) {
  ApplyForce(mMass * g);
  mVelocity += mAccelaration * dt;
  mPosition += position * dt;
  mAcceleration.Set(0f, 0f);
}

RigidBody::ApplyForce(Vector2 f) {
  mAccelaration += f * 1f/mMass;
}
```

Atrito

Também é muito comum implementar uma **força de atrito**, para parar um objeto quando outras forças não estão mais atuando.

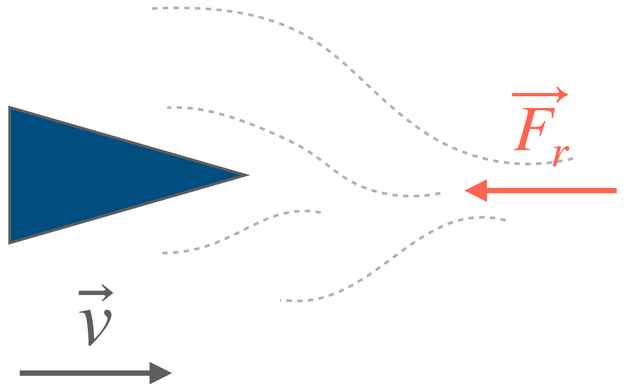



```
float u = 0.01;
float N = 1.0;
float frictionMag = u * N;

Vector2 friction = (-1) * vel;
friction.Normalize();
friction *= frictionMag;
```

Resistência do meio

A mesma ideia se aplica para parar um objeto que não está em contato com uma superfície, mas sobre **resistência do meio**, como do ar ou de um fluído.



$$\vec{F}_r = -\frac{1}{2}\varrho\,||v||^2 A C_d \hat{v}$$

```
float rho = 1.0;
float vLen = vel.Length();
float dragLen = rho * vLen * vLen;
Vector2 drag = (-1) * vel;
drag.Normalize();
drag *= dragLen;
```

- ϱ: densidade do meio
- $||v||^2$: comprimento do vetor velocidade
- ~~A: área frontal do objeto~~
- ~~C_d: coeficiente de resistência~~
- $\hat{v}$: direção do vetor velocidade

Próxima aula

A7: Resolução de Colisão

- Geometrias de colisão
- Detecção de colisão
  - Circunferência vs. Circunferência
  - AABB vs. AABB
- Resolução de Colisaão
- Otimização de Colisão