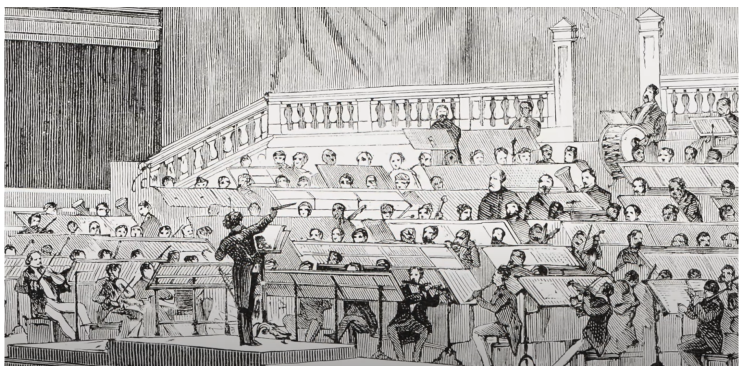
Gravity Demo

Martin N. Håvardsen

October 10, 2025

bildet opp

1 Generell protokoll



Prosesson for simuleringen per objekt skal gå som følger:

utdatert

- tegn
- kalkuler akselerasjon
- kalkuler fart
- kalkuler posisjon

2 Objekt

2.1 Egenskaper

alle objekt skal ha følgende egenskaper:

- posisjon
- fart
- akselerasjon
- masse

„så vil"

2.2 Teori for å kalkulere akselerasjon

For et objekt A med masse m_A og distansen r til et objekt B; Kraften G_B som B påfører A vil være gitt ved følgende:

$$G_B = \gamma \frac{m_A m_B}{r_B^2} \tag{2.1}$$

Gravitasjonskonstanten, γ, vil være en svært liten konstant.

Fra Newtons andre lov får vi at akselerasjonen til objekt A, $\vec{a} = \frac{\overrightarrow{\Sigma F}}{m_A}$. Siden $\overrightarrow{\Sigma F} = \vec{G}_B + \vec{G}_C + \ldots$, for gravitasjonskrefter fra andre objekt B, C osv. $\overrightarrow{\Sigma F}$ kjem berre til å bestå av gravitasjonskrefter i denne simuleringen.

$$\vec{a} = \frac{\hat{G}_B \cdot \gamma \frac{m_A m_B}{r_B^2} + \hat{G}_C \cdot \gamma \frac{m_A m_C}{r_C^2} + \ldots}{m_A} \tag{2.2}$$

$$\vec{a} = \gamma \left(\frac{\hat{G}_B \cdot m_B}{r_B^2} + \frac{\hat{G}_C \cdot m_C}{r_C^2} \right) \tag{2.3}$$

$\hat{G}_B$ vil alltid være rettet mot B, fra A.

$$\hat{G}_B = \frac{\vec{s}_B - \vec{s}_A}{r} \tag{2.4}$$

2.3 Metode for å kalkulere akselerasjon

1. Finn alle retningsvektorer for gravitasjonskreftene. Disse skal oppbevares i en array med en tilsvarende array med masse-verdier. utdatert

2. Regn ut $\vec{a}$

Her er hvordan dette ble implementert:

```python
def update_acceleration(self, all_objects):
    """
    Utfører metode for å kalkulere
    akselerasjon fra seksjon 2.3
    """
    gm = pygame.Vector2(0.0,0.0)
    for obj in all_objects:
        vec=obj.s-self.s
        length=vec.length()
        if length!=0:
            g_hat = vec/length
            gm=gm + g_hat*obj.m/(length**2)
    self.a = gm*GAMMA
```

`obj` her er et objekt fra `PhysObj`-klassen som har et objekt sin masse, posisjon, fart og akselerasjon. Denne funksjonen utføres en gang per *frame* for hvert objekt.

3 Perspektiv og koordinatsystem

3.1 Mål

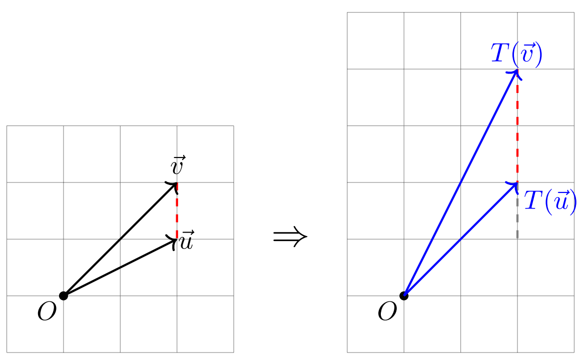
Simuleringen skal ha kamerafunksjonalitet. Man skal kunne bevege koordinatsystemet/rutenettet med musen — bevege *kameraet*. I tillegg skal det være mulig å forstørre rutenettet — *zoome*.

3.2 Teori

?

"dimensjoner"

Planet som objektene tilhører i, er kartesisk og har to ~~retninger~~. Disse retningene, $\hat{i}, \hat{j}$, kan — skalert, kombineres til å få hva som helst punkt på planet.
Koordinaten i vinduet som objektet blir tegnet i er ikke nødvendigvis lik koordinaten på planet. Hvordan transformeringen av romkoordinaten til vinduskoordinat skjer kan variere. Relevante faktorer her er *kameraet* sin posisjon, eventuell *zoom* og *rotasjon*. Her ~~skal vi ignorere~~ rotasjon. "ignorer"
Vi har da at transformeringen $T : \mathbb{R}^2 \to \mathbb{R}^2$,



Before "Før" After "Etter"

Figure 1: Transformeringen av $\vec{v}$ og $\vec{u}$ under T

$$T(\vec{v}) = k\vec{v} + \vec{c} \tag{3.1}$$

Musehjulet skal da forandre k fra (3.1), dette vil forstørre/forminske rutenettet.
Merk at $k \in \langle 0, \infty \rangle$
$\vec{c}$ fra (3.1) er koordinaten på planet som tilsvarer `(0,0)` i vinduet. Vi finn også den inverse transformeringen ved: "finner"

"$T^{-1}(\vec{v})$"

$$T^{-1}(\vec{v}) = \frac{\vec{v} - \vec{c}}{k} = \frac{1}{k}\vec{v} - \frac{1}{k}\vec{c} \tag{3.2}$$

T^{-1} vil da være koordinaten i vinduet som tilsvarer koordinaten på planet.

3.3 Metode