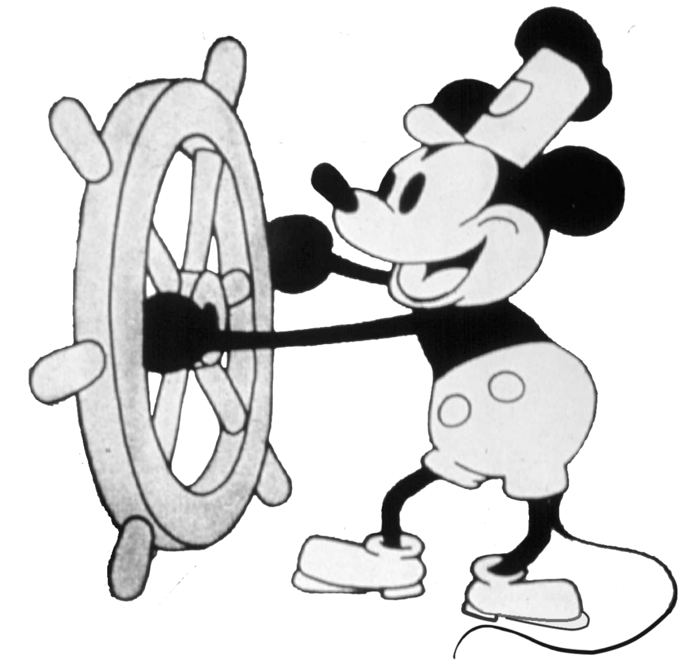


CSCI 461: Computer Graphics

Middlebury College, Fall 2025

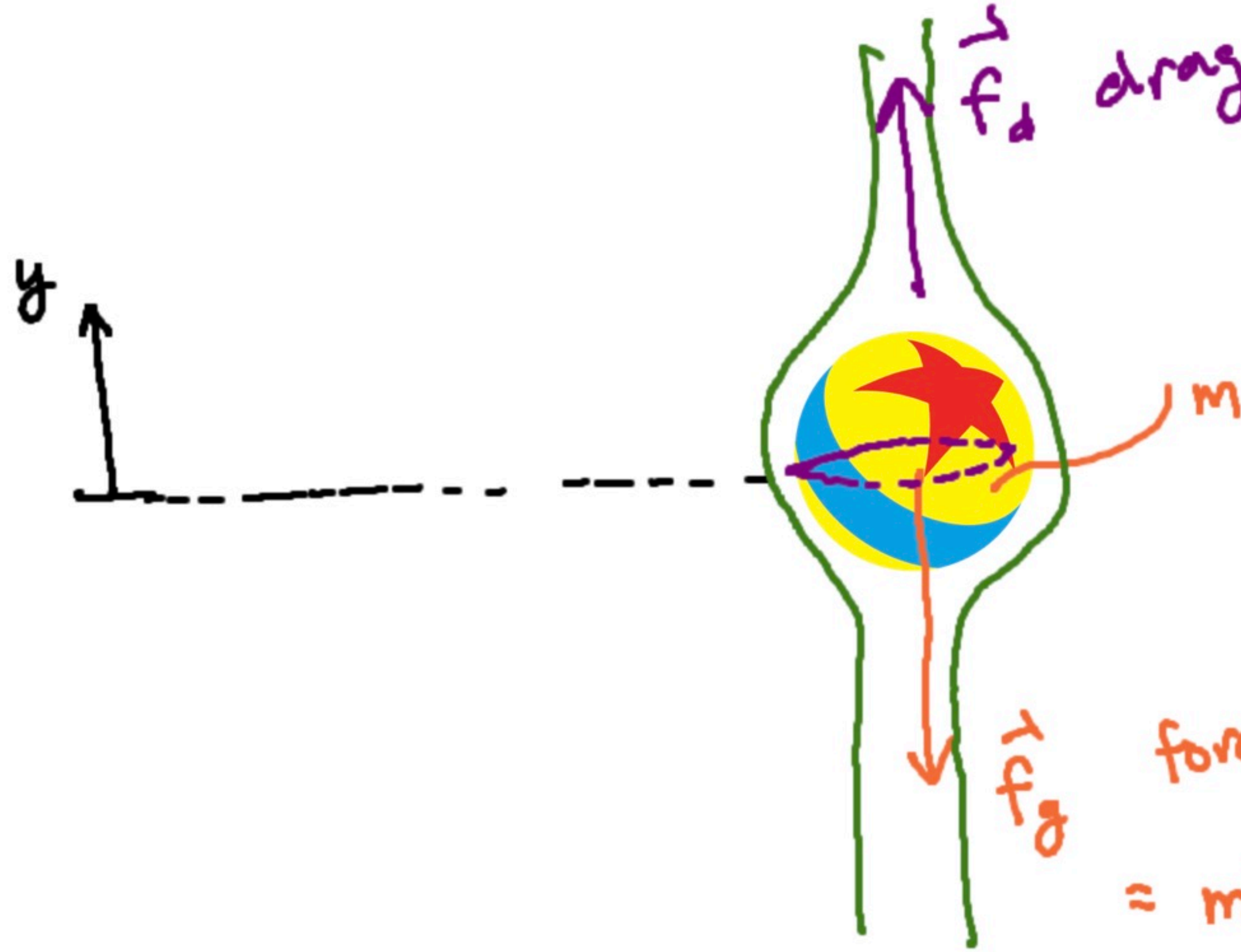
Lecture 11: Particle Animation

In Lab 1, we prescribed motion using curves.

By the end of today's lecture, you will be able to:

- implement **Euler's method** for updating the **position** and **velocity** of particles,
- **draw** particles as either squares or using a sprite,
- use **transform feedback** to **ping-pong** updated position and velocity data during an animation.

Revisiting the Pixar ball from the first lab.



$\vec{f}_d$ drag force

density of air

$$= -\frac{1}{2}\rho A c_d \|\vec{v}\|^{\cancel{2}} \vec{v}$$

$$\cancel{\|\vec{v}\|}$$

$$= -\frac{1}{2}\rho A c_d \|\vec{v}\| \vec{v}$$

$$\frac{kg}{m^3} m^2 \frac{m}{s} \frac{m}{s} = kg \frac{m}{s^2}$$

$\vec{f}_g$ force of gravity

$$= m\vec{g} \qquad \vec{g} = \begin{bmatrix} 0 \\ -9.81 \\ 0 \end{bmatrix} m/s^2$$

$$kg \frac{m}{s^2}$$

Let's just use a computer...

$$\vec{v}^k = \frac{\Delta p}{\Delta t} = \frac{\vec{p}^{k+1} - \vec{p}^k}{\Delta t}$$

$$\vec{a}^k = \frac{\Delta v}{\Delta t} = \frac{\vec{v}^{k+1} - \vec{v}^k}{\Delta t}$$

$$\rightarrow \vec{V}^{k+1} = \vec{V}^k + \Delta t\, \vec{a}^k$$

$$\vec{V}^{k+1} = \vec{V}^k + \Delta t \left(\frac{\Sigma \vec{f}}{m} \right)$$

$$\Sigma \vec{f} = m\vec{a} = \vec{f}_d + \vec{f}_g$$

$$\vec{a} = \frac{\Delta \vec{v}}{\Delta t} \quad \rightarrow \quad \vec{a}^k = \frac{\Sigma \vec{f}}{m}$$

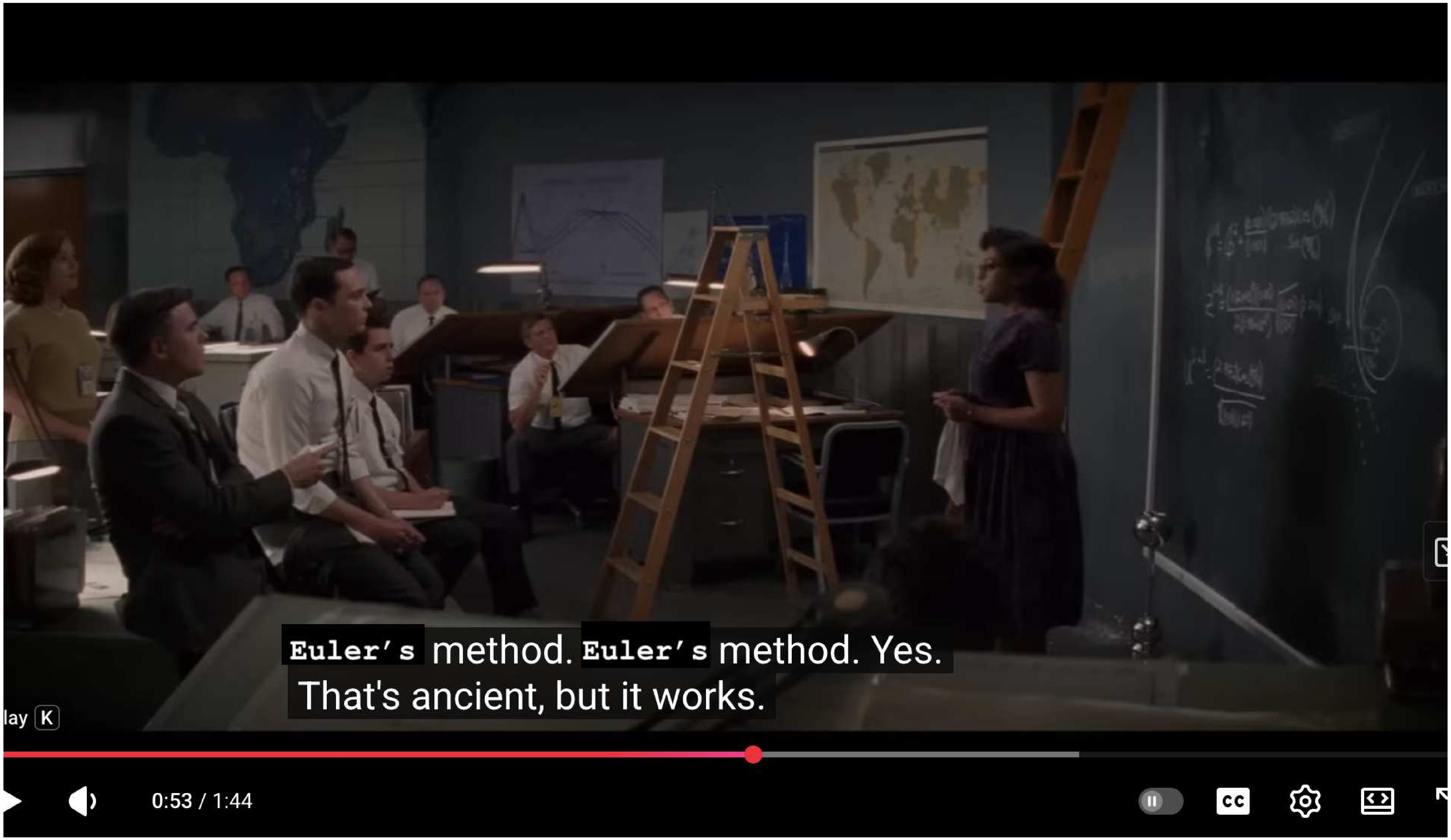
$\uparrow \vec{f}_d$

$\downarrow \vec{f}_g$

$$\vec{p}^{k+1} = \vec{p}^k + \Delta t\, \vec{V}^k$$

k: frame in animation

Euler's method was used in *Hidden Figures*.

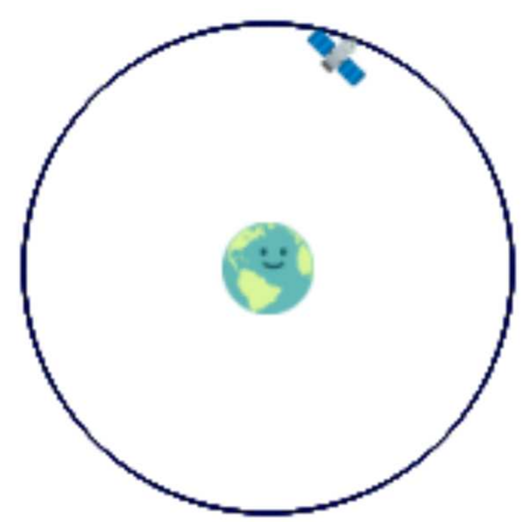


Accuracy of Euler's method depends on time step Δt.

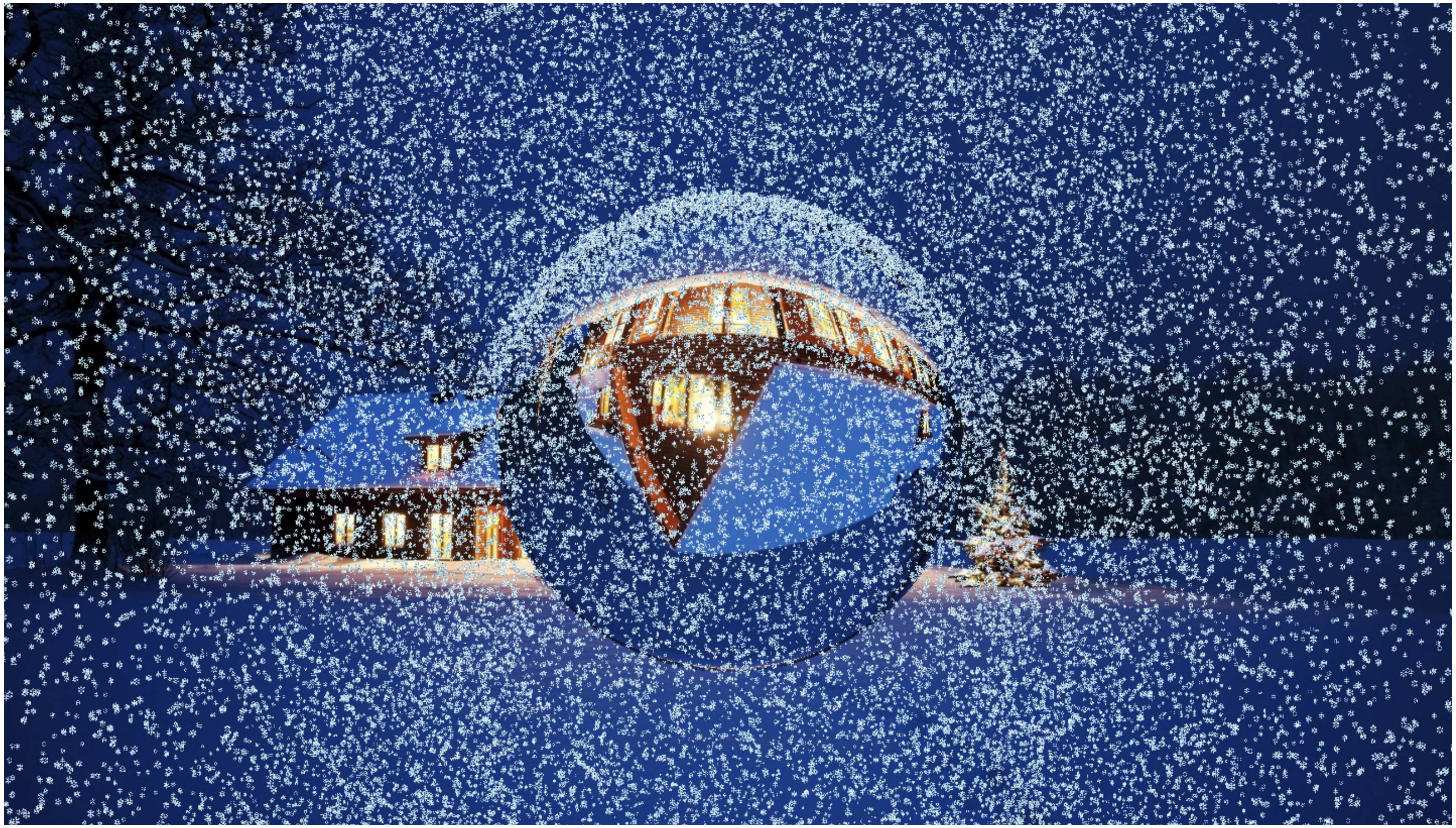
demo: https://philipclaude.github.io/csci461f25/chapter11

☑ Euler (red) ☐ Runge-Kutta (blue) ☑ Analytic (black)

run



Our goal: animate thousands or millions of particles!

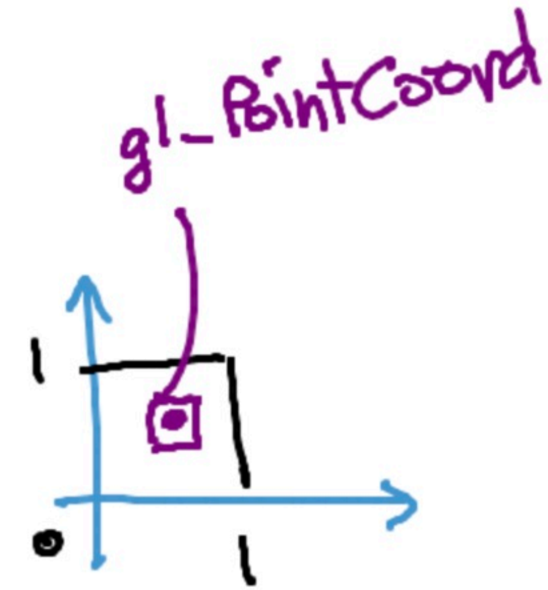


Draw particles with `gl.drawArrays` and `gl.POINTS`.

```
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
gl.vertexAttribPointer(a_Position, 3, gl.FLOAT, false, 0, 0);
gl.drawArrays(gl.POINTS, 0, nParticles);
```

new

new




```
attribute vec4 a_Position;

uniform mat4 u_ProjectionMatrix;
uniform mat4 u_ViewMatrix;

void main() {
  gl_Position = u_ProjectionMatrix * u_ViewMatrix * vec4(a_Position, 1);
  gl_PointSize = 50.0 / gl_Position.w;
}
```

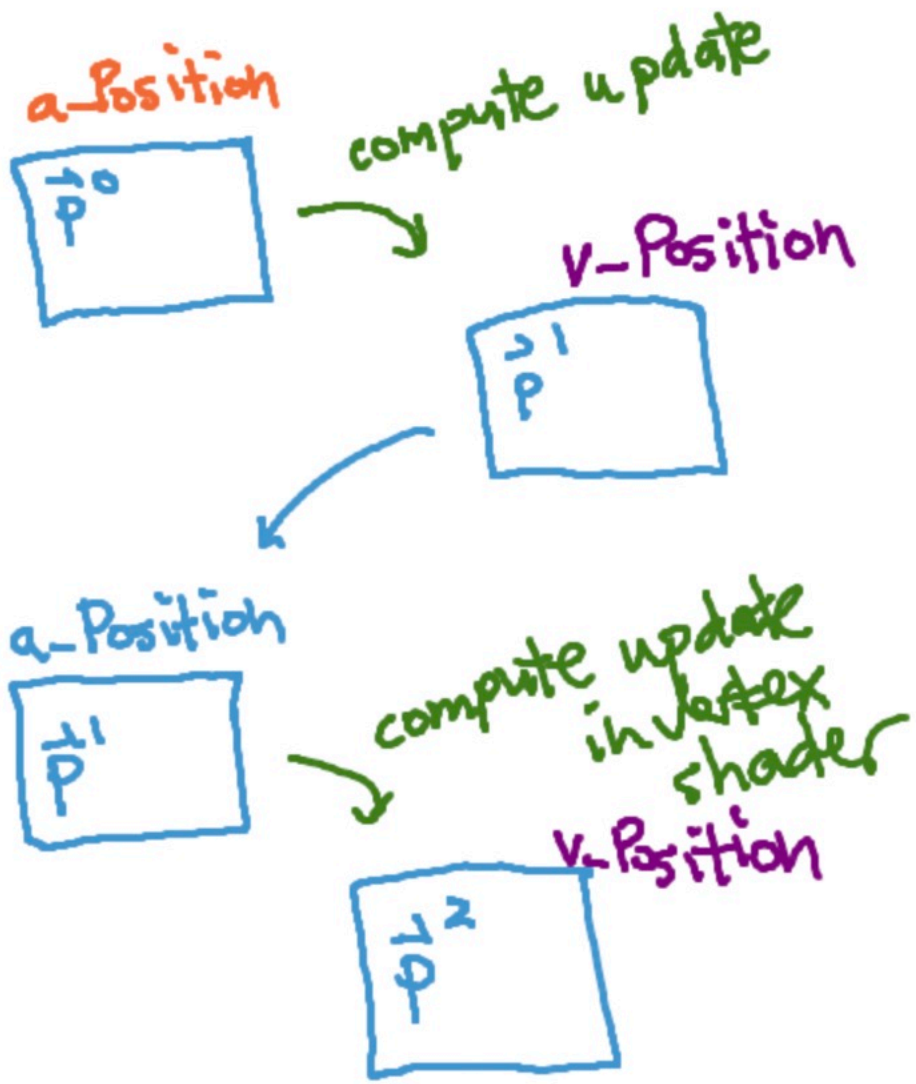
new

```
uniform sampler2D tex_Sprite;

void main() {
  gl_FragColor = texture2D(tex_Sprite, gl_PointCoord);
  //gl_FragColor = vec4(1, 1, 1, 1);
}
```

Complete particle animation might look like this:

```
// initialize particle positions and velocity
let nParticles = 1000;
let position = new Array(nParticles * 3);
// initialize position and velocity with random or known data...

// assume gl is some WebGLRenderingContext
const draw = (position) {
  let positionBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(position), gl.STATIC_DRAW);
  gl.drawArrays(gl.POINTS, 0, nParticles);
}

const mass = 1; // some mass in kg
const nSteps = 1000;
const tFinal = 10;
const deltaT = tFinal / nSteps;

for (let k = 0; k < nSteps; k++) {
  draw(position);

  // for each particle, calculate the update
  for (let i = 0; i < nParticles; i++) {
    const ak = vec3.fromValue(0, -9.81, 0); // only gravity in this example
    for (let d = 0; d < 3; d++) {
      const vNext = velocity[3 * i + d] + deltaT * ak[d];
      const pNext = position[3 * i + d] + deltaT * velocity[3 * i + d];

      velocity[3 * i + d] = vNext;
      position[3 * i + d] = pNext;
    }
  }
}
```

We'll use *transform feedback* to capture updated position & velocity to a buffer and swap buffers at every iteration.



swap buffers at the end of each time step

Example: use transform feedback to animate particles.

- Part 1: declare varyings to capture.
- Part 2: create buffers for transform feedback.
- Part 3: calculate updated position and velocity in vertex shader.
  - Today, we'll use $\vec{p}^{k+1} = \vec{p}^{k} + \vec{v}^{0}\Delta t$ with a *constant* velocity $\vec{v}^{0}$.
  - You will actually use Euler's method to update velocity in the lab on Thursday.
  - Only really working with position in this example (to practice with transform feedback).
- Part 4: draw and capture updated position in feedback buffer, then swap buffers.

Everything you need to include velocity updates is in the reading.

Recap of particle animation (with transform feedback).

$$\vec{v}^k = \frac{\Delta p}{\Delta t} = \frac{\vec{p}^{k+1} - \vec{p}^k}{\Delta t} \quad \rightarrow \quad \vec{p}^{k+1} = \vec{p}^k + \Delta t \, \vec{v}^k,$$

$$\vec{a}^k = \frac{\Delta v}{\Delta t} = \frac{\vec{v}^{k+1} - \vec{v}^k}{\Delta t} \quad \rightarrow \quad \vec{v}^{k+1} = \vec{v}^k + \Delta t \, \vec{a}^k.$$

Summary

- Use Euler's method to update motion,
- Be careful with time step Δt,
- Use transform feedback to keep everything in the GPU,
- Can even use transform feedback for GPGPU programming,
- More complicated when there are interparticle forces (e.g. springs) - after Thanksgiving!