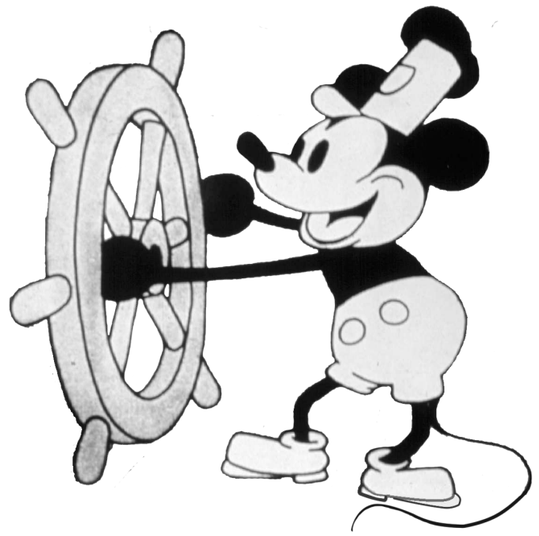


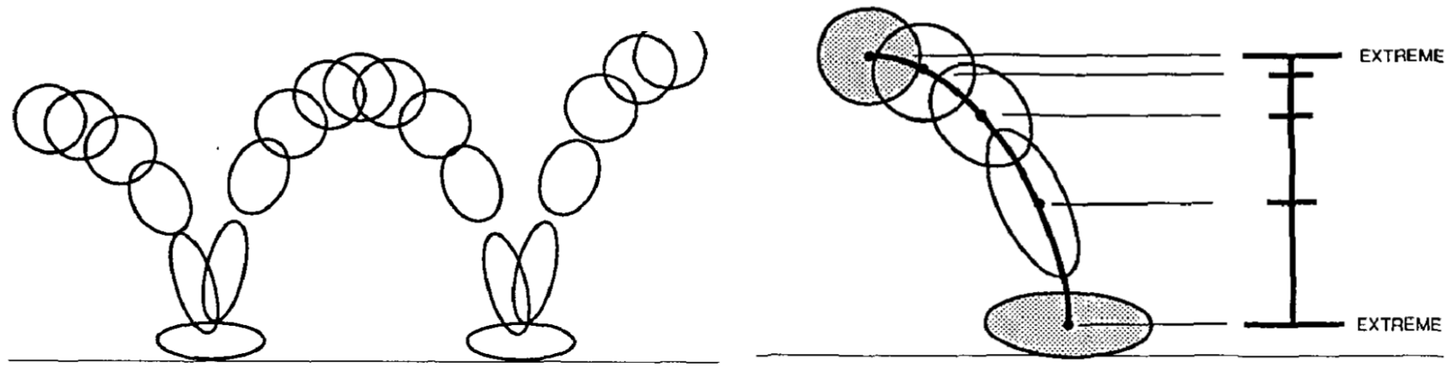
CSCI 461: Computer Graphics

Middlebury College, Fall 2025

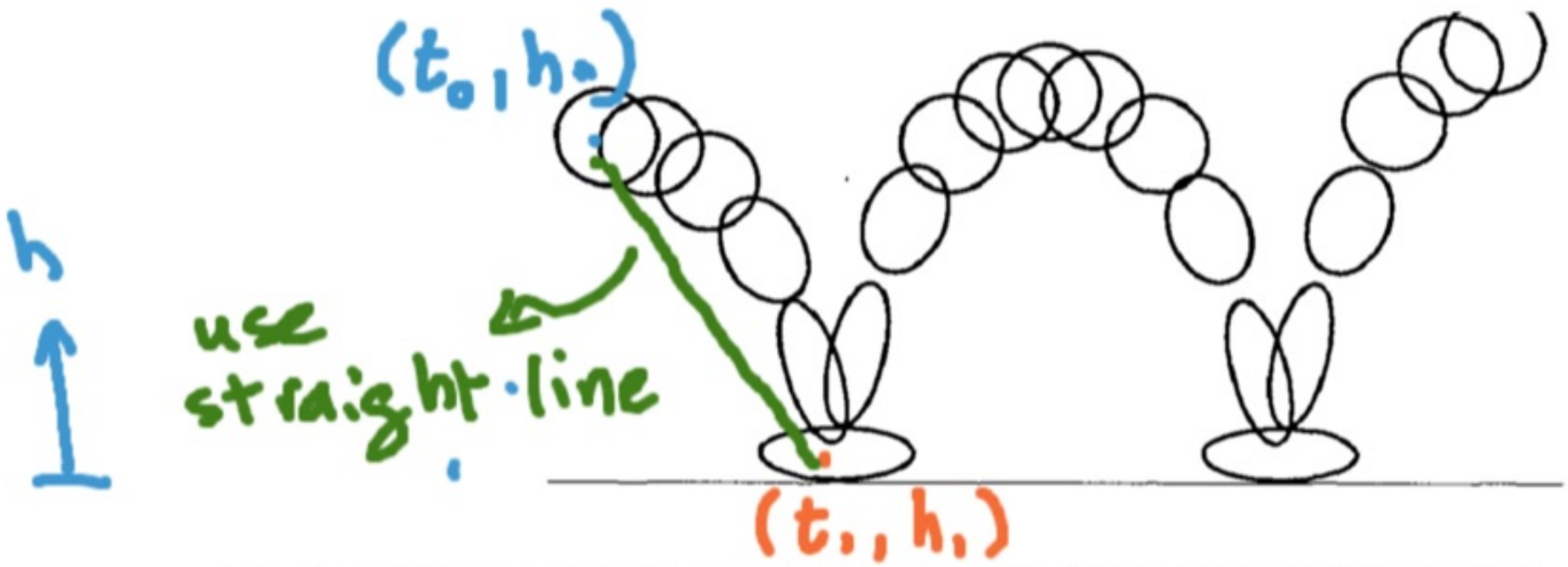
Lecture 2B: Linear Algebra (Matrices)

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

- **multiply a matrix with a vector** (and use `glMatrix vecN.transformMatN`),
- **multiply a matrix with another matrix** (and use `glMatrix matN.multiply`),
- **invert** a 2x2 matrix (by hand) and 3x3 matrices with `glMatrix matN.invert`,
- **interpolate** parameters defined at chosen frames with either lines or cubic curves,
- apply **de Casteljau's algorithm** to evaluate and render a cubic Bézier curve.



Animation artists specify scene parameters at selected frames.
Figuring out what happens in between frames is called inbetweening.



How to inbetween with a computer/mathematically? **Let's try interpolation.**

$h(t) = at + b$

2 unknowns: a, b

$h_0 = a\,t_0 + b$
$h_1 = a\,t_1 + b$

2 equations

This leaves us with two equations and two unknowns ($\bar{a}$ and $\bar{b}$).

$h_0 = a t_0 + b$

$h_1 = a t_1 + b$

$\vec{h} = a\vec{t} + b\vec{1}$

introduce: $\vec{h} = \begin{bmatrix} h_0 \\ h_1 \end{bmatrix}$ $\vec{t} = \begin{bmatrix} t_0 \\ t_1 \end{bmatrix}$

$\vec{1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

can also write as: $\begin{bmatrix} h_0 \\ h_1 \end{bmatrix} = \begin{bmatrix} t_0 & 1 \\ t_1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$

matrix A (2x2) $\quad \vec{c}$

matrix-vector multiplication.

$\vec{h} = A\vec{c}$

solve for $\vec{c}$

To solve this equation for x, we can multiply both sides by a^{-1}.

$$ax = b$$

$$a^{-1}ax = a^{-1}b$$

$a^{-1}a = 1$

$$x = a^{-1}b$$

We can use the same idea with matrices: multiply both sides by the "inverse" of our matrix.

we had $A\vec{c} = \vec{h}$

$$A^{-1}A\vec{c} = A^{-1}\vec{h}$$

$$I\vec{c} = A^{-1}\vec{h}$$

There's a formula for the inverse of 2x2 matrices!

$$\mathbf{A} = \begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix}$$

$$A^{-1} = \frac{1}{a_{00}a_{11} - a_{01}a_{10}} \begin{bmatrix} a_{11} & -a_{01} \\ -a_{10} & a_{00} \end{bmatrix}$$

$$= \frac{1}{d} \begin{bmatrix} a_{11} & -a_{01} \\ -a_{10} & a_{00} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{a_{11}}{d} & -\frac{a_{01}}{d} \\ -\frac{a_{10}}{d} & \frac{a_{00}}{d} \end{bmatrix}$$

Matrix-matrix multiplication.

$A^{-1}A$

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \quad B = \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$$

$$AB = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} = \begin{bmatrix} A\vec{b}_0 & A\vec{b}_1 \end{bmatrix}$$

A $\quad \vec{b}_0 \quad \vec{b}_1$

$$= \begin{bmatrix} (a_{00}b_{00} + a_{01}b_{10}) & (a_{00}b_{01} + a_{01}b_{11}) \\ (a_{10}b_{00} + a_{11}b_{10}) & (a_{10}b_{01} + a_{11}b_{11}) \end{bmatrix}$$

Exercise: calculate what $\mathbf{A}^{-1}\mathbf{A}$ is equal to.

$$\mathbf{A} = \begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix}, \quad \mathbf{A}^{-1} = \frac{1}{a_{0,0}a_{1,1} - a_{0,1}a_{1,0}} \begin{bmatrix} a_{1,1} & -a_{0,1} \\ -a_{1,0} & a_{0,0} \end{bmatrix}.$$

Assign each group member one entry.

Let's get back to our goal of linear interpolation.

At $t = 0.5$ seconds, let the ball have a height of 0.25 (meters) and at $t = 0.75$ seconds, the height of the ball is 2 meters. Using linear interpolation, determine an expression for the height as a function of t.

1. Construct the matrix $\mathbf{A}$ and right-hand-side vector $\vec{b}$.
2. Compute the matrix inverse $\mathbf{A}^{-1}$.
3. Compute $\vec{c}$ from the matrix-vector $\vec{c} = \mathbf{A}^{-1}\vec{b}$.

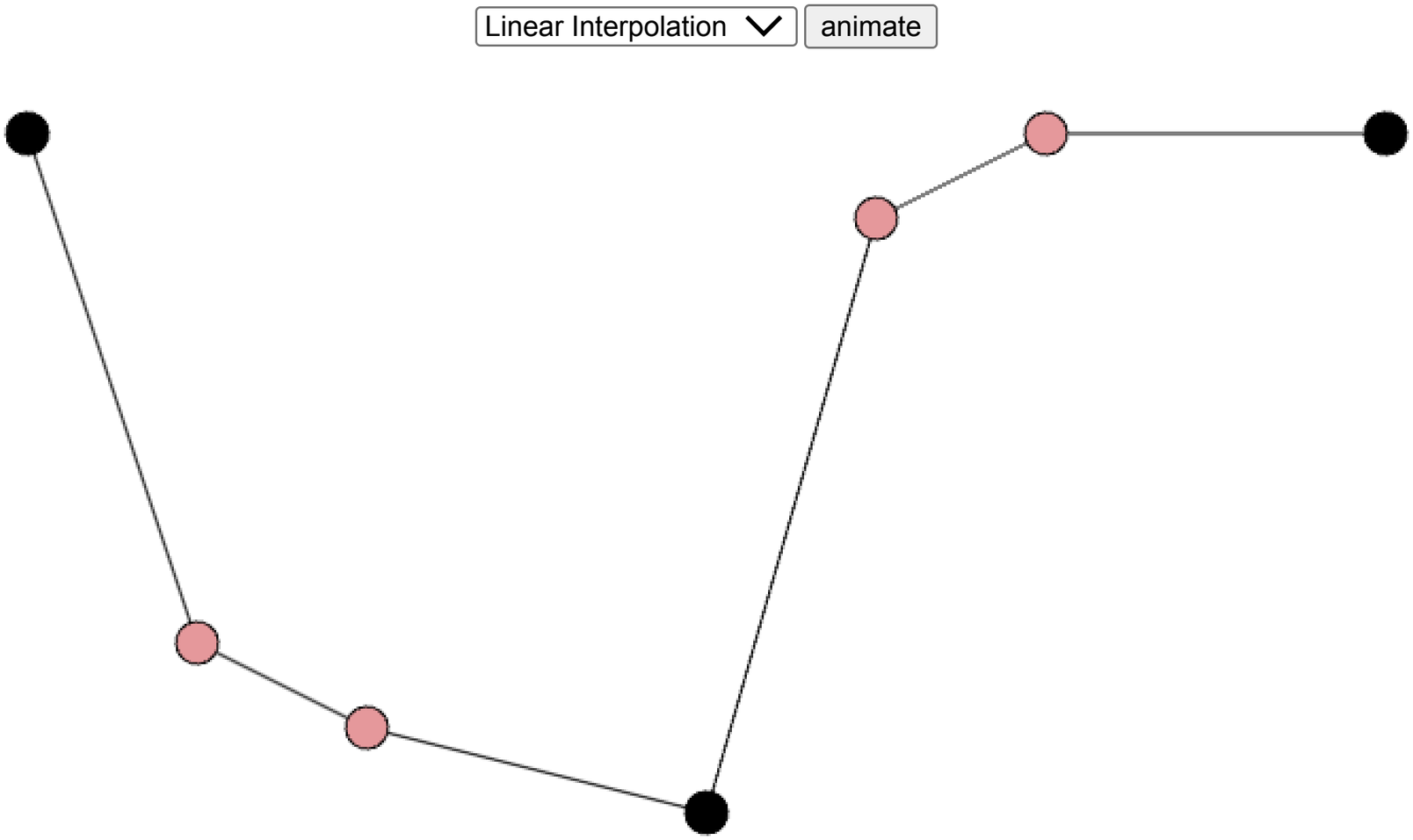
Using `glMatrix` to do this for us...

At $t = 0.5$ seconds, let the ball have a height of 0.25 (meters) and at $t = 0.75$ seconds, the height of the ball is 2 meters. Using linear interpolation, determine an expression for the height as a function of t.

```
let rhs = vec2.fromValues(0.25, 2);
let A = mat2.fromValues(0.5, 0.75, 1, 1);
let Ainv = mat2.invert(mat2.create(), A);
let c = vec2.transformMat2(vec2.create(), rhs, Ainv);
console.log(c); // [7, -3.25]

// we can also check that A * Ainv is equal to the identity matrix
let I = mat2.multiply(mat2.create(), A, Ainv);
console.log(I); // [1, 0, 0, 1] (again printed in column-major order)
```

BUG ALERT: `glMatrix` reads entries in COLUMN-MAJOR order!

Putting this together for several keys, we get an animation that looks like this (see the demo in the notes).



Our animation is kind of choppy. Let's try to interpolate the keys using a *cubic* curve instead.

We'll assume a curve that looks like:

$$h(t) = a\,t^3 + b\,t^2 + c\,t + d.$$

So we need to figure out a, b, c, and d. It's the same procedure as before - we're just working with $4d$ vectors and 4x4 matrices.

$$\begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix} = \begin{bmatrix} t_0^3 & t_0^2 & t_0 & 1 \\ t_1^3 & t_1^2 & t_1 & 1 \\ t_2^3 & t_2^2 & t_2 & 1 \\ t_3^3 & t_3^2 & t_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$\vec{h} = \quad \vec{t} \quad \vec{a}$$

Exercise: use `glMatrix` to interpolate the following (time, height) keys, and then evaluate the curve at $t = 0.75$

time	height
0	1
0.25	0.25
0.5	0.125
1	0

$$\vec{h} = \begin{bmatrix} t_0^3 & t_0^2 & t_0 & 1 \\ t_1^3 & t_1^2 & t_1 & 1 \\ t_2^3 & t_2^2 & t_2 & 1 \\ t_3^3 & t_3^2 & t_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

Useful functions:

- `vec4.fromValues`
- `mat4.fromValues`
- `vec4.transformMat4`

Remember that `glMatrix` reads entries in column-major order.

Look up the `glMatrix` documentation!

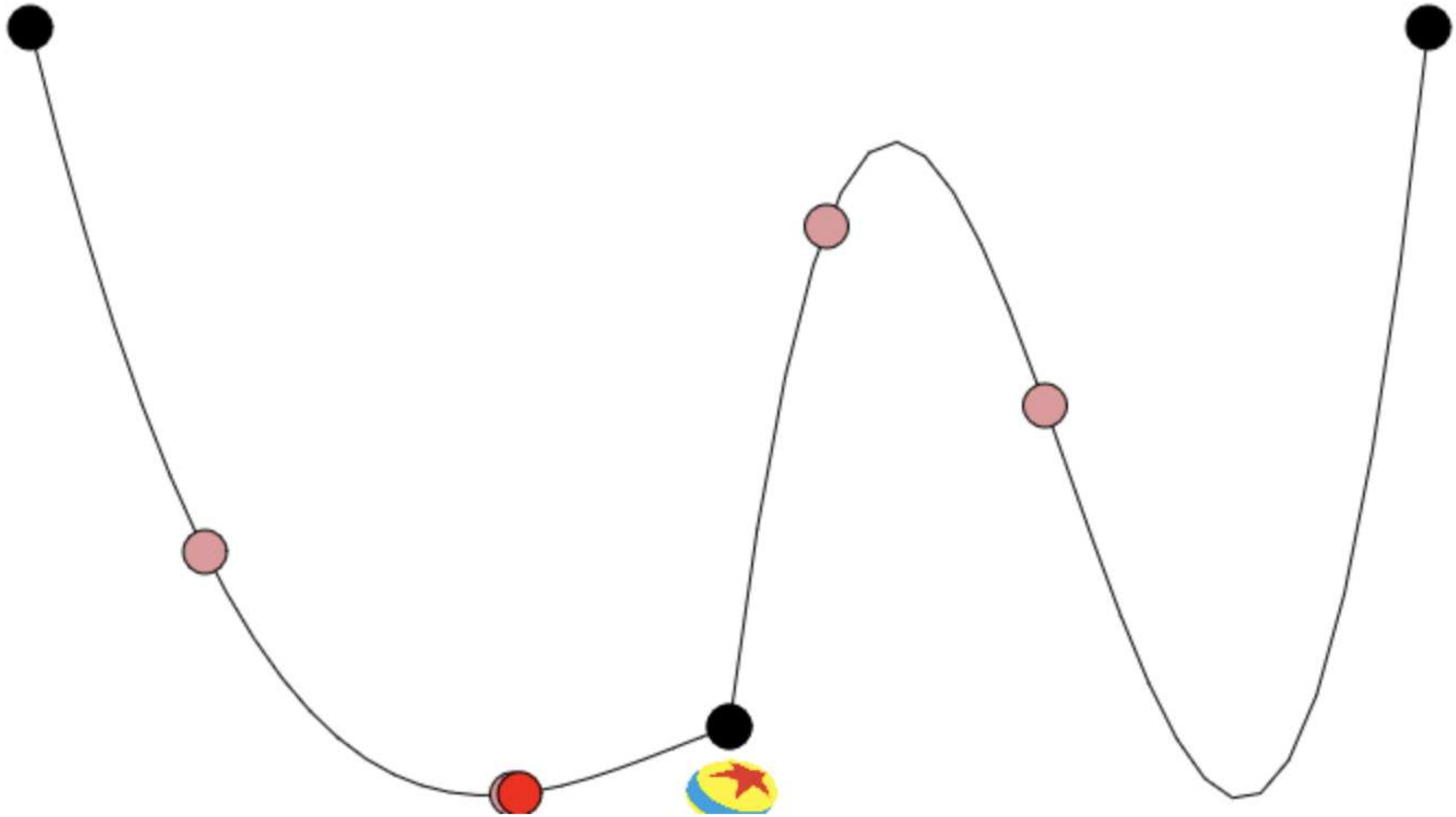
`Math.pow(t0, 3)`

$$h(t) = at^3 + bt^2 + ct + d$$

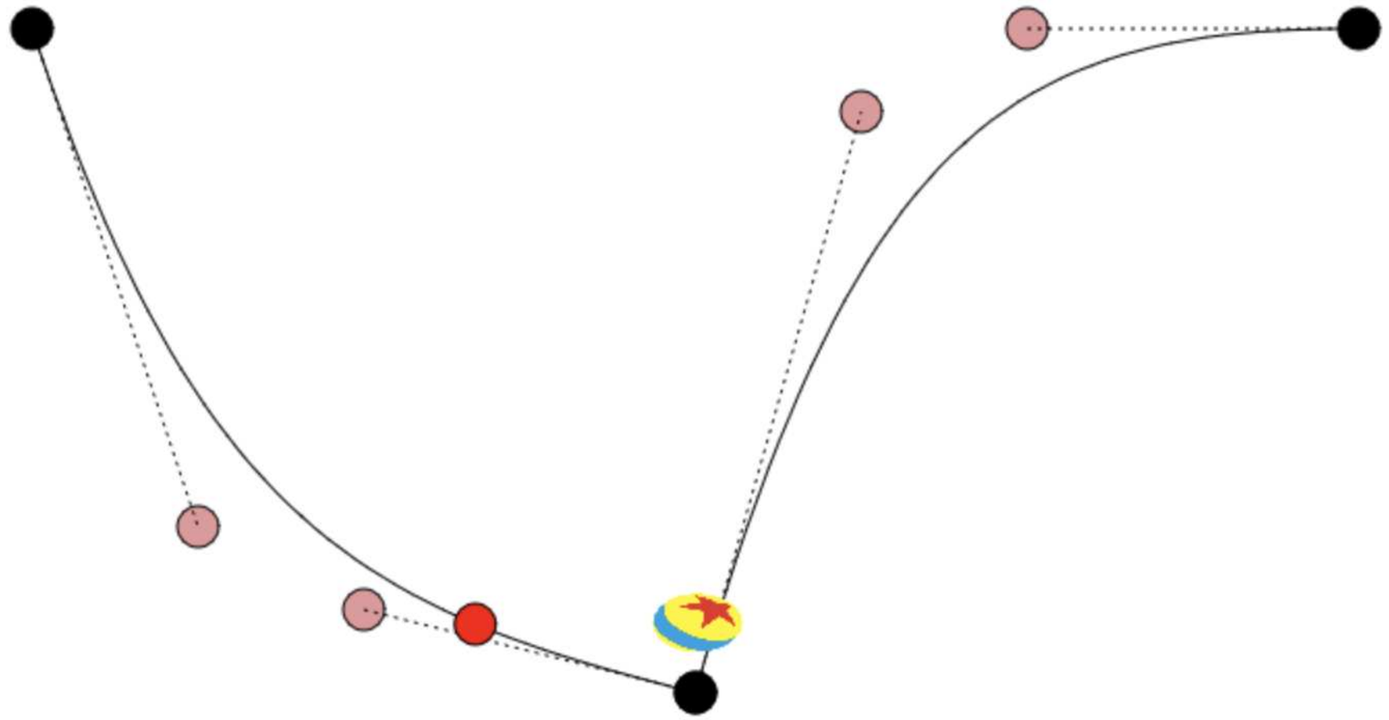
Possible implementation with `glMatrix`

```
// given data
const t = [0, 0.25, 0.5, 1];
const h = [1, 0.25, 0.125, 0];

// setup matrix
let A = mat4.create();
for (let i = 0; i < 4; i++) {
  A[i] = Math.pow(t[i], 3);
  A[i + 4] = t[i] * t[i];
  A[i + 8] = t[i];
  A[i + 12] = 1.0;
}

// compute inverse
const Ainv = mat4.invert(mat4.create(), A);

// solve system of equations
const c = vec4.transformMat4(vec4.create(), h, Ainv);

// determine height at t = 0.75
const time = 0.75;
const time2 = time * time;
let ht = time * time2 * c[0] + time2 * c[1] + time * c[2] + c[3];
```

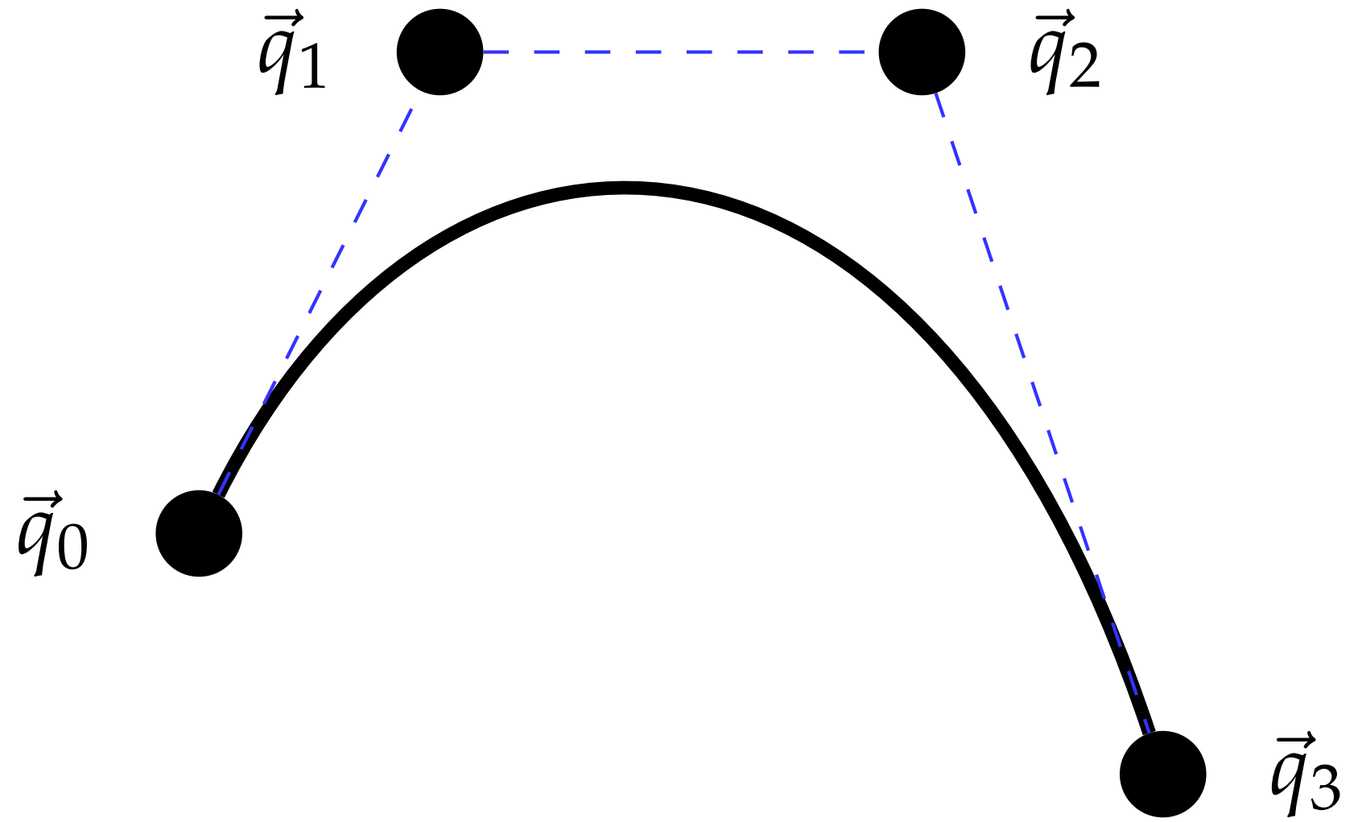
But interpolation is still not great: the curve might not prescribe the motion we want.



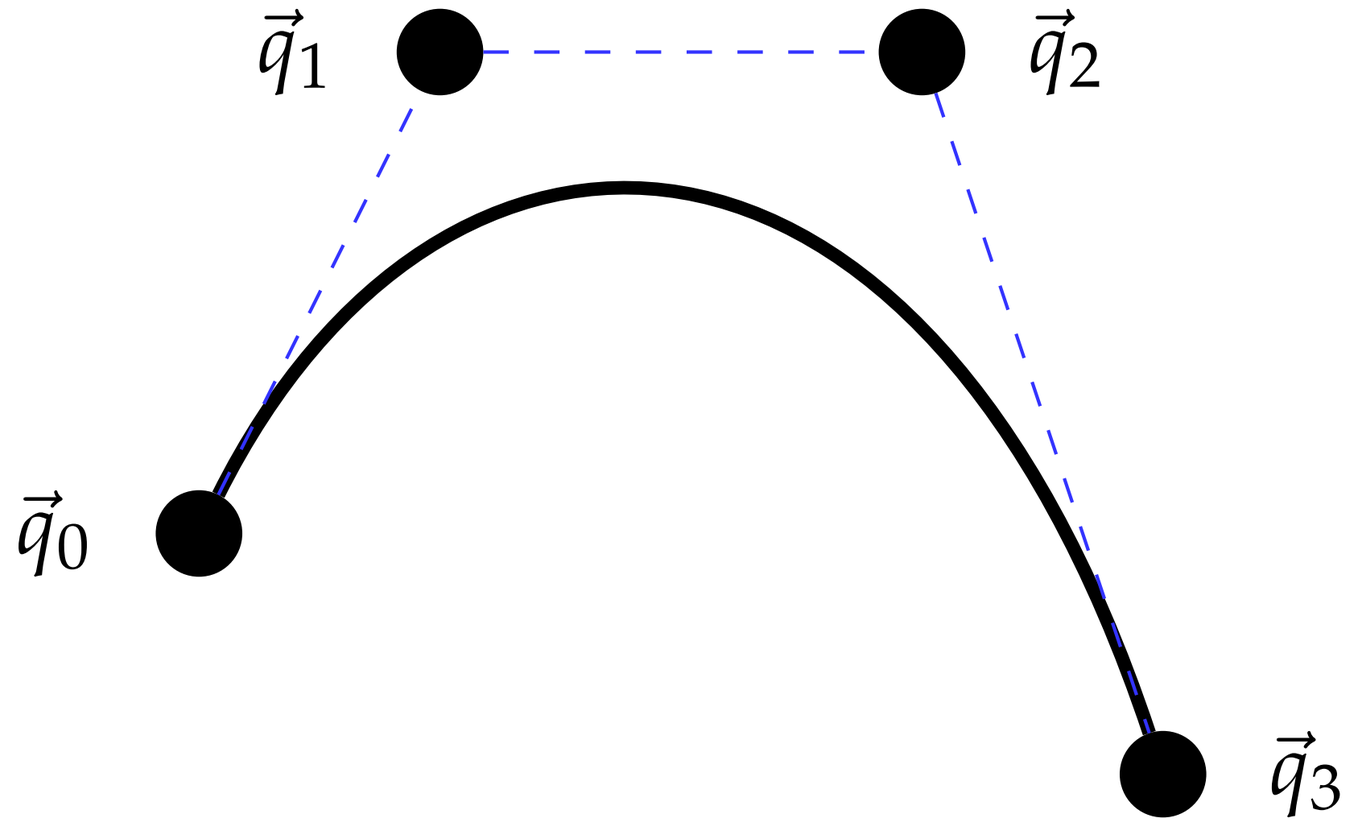
Instead we can use Bézier curves: let the two interior points allow us to control the slope of the curve at the endpoints.



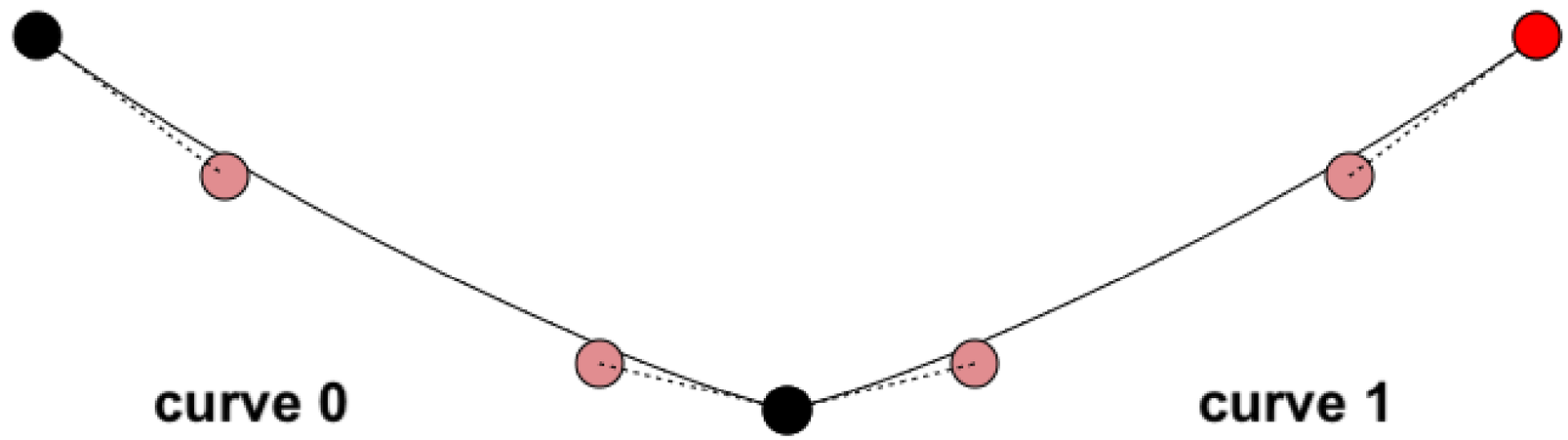
Using de Casteljau's algorithm to evaluate a cubic Bézier curve at some arbitrary t.



Using de Casteljau's algorithm to render a cubic Bézier curve.



Two curves in Lab 1: one for downwards motion and one for upwards motion.



Summary

- First lab on Thursday! Complete Pre-Lab beforehand (linked on calendar).
  - You'll implement your own cubic curve interpolator and evaluator.
  - You'll implement your own cubic Bézier curve evaluator and renderer.
- Remember `glMatrix` assumes entries are ordered in column-major order.
- Matrix-matrix multiplication with `C = mat4.multiply(C, A, B)` to compute $\mathbf{C} = \mathbf{AB}$ (or `mat2.multiply`, `mat3.multiply`).
- Matrix-vector multiplication with `b = vec4.transformMat4(b, x, A)` to compute $\vec{b} = A\vec{x}$ (or `vec2.transformMat2`, `vec3.transformMat3`).
- Matrix inverse with `Ainv = mat4.invert(Ainv, A)` (or `mat2.invert`, `mat3.invert`).
- **Office hours :** (please come say hi!)
  - Mondays: 10am - 11am
  - Tuesdays: 11am - 11:30pm
  - Thursdays: 2:30pm - 4:30pm