CSC 317

Tutorial: Mass Spring System

signed_incidence_matrix_dense

- For each edge, set A's matrix value
- E is (# edges) x 2 matrix
  - Each row describes an edge containing two vertices
  - ie, E(e,0) and E(e,1) give the two vertex indices for edge e
  - Take E(e,0) to be vertex i and E(e,1) to be vertex j
- A is (# edges) x (# vertices) matrix

$$\mathbf{A}_{ek} = \begin{cases} +1 & \text{if } k = i \\ -1 & \text{else if } k == j \\ 0 & \text{otherwise.} \end{cases}$$

fast_mass_springs_precomputation_dense

- Need to find the action of Q inverse
- Assemble Q then prefactor it (refer to Eigen::LLT)
- M is a diagonal matrix of masses
- signed_incidence_matrix gives A
- prefactorization.compute(Q) does the required "inverse"

$$\mathbf{M}_{ij} = \begin{cases} m_i & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathbf{Q} := (k\mathbf{A}^\top \mathbf{A} + \frac{1}{\Delta t^2}\mathbf{M}) \in \mathbf{R}^{n \times n}$$

\+ pinned vertex penalty term (quadratic)

- Also fill in:
  - r: list of edge lengths
  - C: selection matrix (=1 for every pinned vertex)

Fast Mass Springs

This leads to a straightforward "local-global" iterative algorithm:

Step 1 (local): Given current values of p determine d_ij for each spring.

Step 2 (global): Given all d_ij vectors, find positions p that minimize quadratic energy

Step 3: if "not satisfied", go to Step 1.

fast_mass_springs_step_dense

- Find the new p (Unext)
- Note the differences between:
  - Uprev = positions at t-dt
  - Ucur = positions at t
  - V = positions at rest
- Careful: do not create a new variable named b
  - Already used for the list of pinned vertices
- Remember that we need 50 iterations of a local-global solve
  - For each iteration:
    - find d (directions), that attempts to preserve current edge length
      - (ie normalize d then multiply by edge lengths e)
    - then find p using $\mathbf{p} = \mathbf{Q}^{-1}\mathbf{b}.$
- Notice that y is constant for each iteration, only b changes (since d changes)
  - ie construct y once before the iterations

$$\mathbf{y} := \frac{1}{\Delta t^2}\mathbf{M}(2\mathbf{p}^t - \mathbf{p}^{t-\Delta t}) + \mathbf{f}^{\text{ext}} \in \mathbf{R}^{n\times 3}$$

\+ pinned vertex penalty term (linear)

$$\mathbf{b} := k\mathbf{A}^\top\mathbf{d} + \mathbf{y} \in \mathbf{R}^{n\times 3}.$$

$$\mathbf{v} = \mathbf{A}\mathbf{p} \leftrightarrow \mathbf{v}_{ij} = \mathbf{p}_i - \mathbf{p}_j.$$

pinned vertices

- Need to differentiate the energy:

$$\frac{w}{2}\,\mathrm{tr}\left((\mathbf{C}\mathbf{p}-\mathbf{C}\mathbf{p}^{\mathrm{rest}})^\top(\mathbf{C}\mathbf{p}-\mathbf{C}\mathbf{p}^{\mathrm{rest}})\right) = \frac{1}{2}\,\mathrm{tr}\left(\mathbf{p}^\top(w\mathbf{C}^\top\mathbf{C})\mathbf{p}\right) - \mathrm{tr}\left(\mathbf{p}^\top w\mathbf{C}^\top\mathbf{C}\mathbf{p}^{\mathrm{rest}}\right) + \mathrm{constant}$$

- Construct C from variable name b (list of pinned vertices)
- Once you have the derivative:
  - The quadratic term gets added to Q (first RHS term; in precomputation)
  - The linear term gets added to b (second RHS term; in step)
- Follow the derivation of Q and b
- Remember input V is the list of rest vertex positions

sparse versions

- Copy-paste dense code and change all the dense matrix datatypes
- All large dense matrices should be converted to sparse
- Vectors stay dense
- Use Eigen's [setFromTriplets]()
    - Use std::vector for constructing the list of triplets
    - ijv.emplace_back(i,j,v); means adding entry with row# i, col# j, and value v
- DO NOT add zero entries!
- DO NOT create a dense matrix then convert it to sparse!
- These defeat the purpose of using sparse matrices, since they will contain explicit zero entries, taking up memory and slowing down matrix multiplies