Final Projects – Due Monday May 12th 11:59 ET

In the final project, each team will propose a project to work on. The project may come from students' research, interests, or the prompts below.

The goal is to apply the design patterns learned throughout the semester to create versatile and interesting code, learn and use efficient computational libraries, and think deeply about the way code is written.

The code you develop in the final project should be understandable and usable by other teams as well. Final project teams may develop one or more prompts below or develop their own research-related system and trade code with other teams throughout the project. Thus, each team may trade code and work with other teams to integrate any developed components and create something awesome, such as a world containing:

- A dam-like shallow-water model,
- Into which drops of water (modeled as meshed mass-springs) are falling,
- Whose motion is controlled by forces such as gravity, air resistance, and wind,
- And which, when they contact the shallow water surface, merge with that surface by adding water.
- Meanwhile, on that water, a perhaps-spherical boat, modeled by a tetrahedral or pressure-surface mesh, is floating,
- Which is being buffeted about by currents,
- Which are displayed as vectors.
- Perhaps there are multiple boats, which can collide—causing one to sink.
- And the water is flowing out of the shallow-water model through a hole.

The purpose of this project is, first, to give you additional experience with physics-based models; and second, to prepare you for the important real-world experience of writing your code for others and integrating your code with others'. Remember that you will be writing code that other teams may want to use and that you may be using other teams' code. Plan ahead. Document your requirements. Provide specification comments for new functions. Write test cases.

Each section below enumerates possible extensions to problems that we have already seen and a few that we have not. These may be treated as starting points and suggestions for creating interesting 3D physical models. A successful final project will be composed of multiple extensions with code integrated and borrowed from other teams, or a completely new related to a research project prompt below.

Prompt 1 – Meshed Mass Spring

In HW2, we used the `Graph` to implement a mass-spring model of a cloth. Using a triangular or tetrahedral `Mesh` instead, more advanced operations can be implemented

- A simple model of force due to wind can be written as

$$\mathbf{f}_{i,\text{air}} = c\left((\mathbf{w} - \mathbf{v}_i) \cdot \hat{\mathbf{n}}_i\right) \hat{\mathbf{n}}_i$$

  where $\mathbf{w}$ is the air velocity, $\mathbf{v}_i$ is the velocity of node i, $\hat{\mathbf{n}}_i$ is the unit surface normal at node i (note that orientation does not matter), and c is some interaction factor. As a good approximation, we can set $\mathbf{w}$ to a constant. A simple way to approximate $\mathbf{n}_i$ is to simply use the neighboring faces:

$$\mathbf{n}_i = \sum_{T_k \in \mathcal{T}(i)} \hat{\mathbf{n}}_k$$

  where $\mathcal{T}(i)$ is the set of incident faces to node i and $\hat{\mathbf{n}}_k$ is the unit normal vector of face T_k. Note that $\hat{\mathbf{n}}_k$ changes with the mesh and must be recomputed at each time step. Thus, an implementation of the mass-spring model with the `Mesh` class could use the node-triangle incidence you implemented in HW3 to compute node normals and forces due to wind.

  Can you represent the mass-spring meshes from HW2 with your `Mesh` class? or do they violate any of the assumptions or representation invariants of your `Mesh`? You may consider improving/generalizing your `Mesh` class or computing the wind forces by face rather than by node.

- We can make other cool things with triangular meshes. Consider using a triangular mesh as the surface of an object. Endowing this mesh with mass-spring physics will probably not give it the best structural integrity and it will act similar to the cloth example in most cases – falling on/through itself and raveling up. Can we make it more stable? Let's fill it with air. From ye olde $PV = nRT$ ideal gas law, the force *on each face* due to the internal air pressure is given by

$$\mathbf{F}_k = P\,|T_k|\,\hat{\mathbf{n}} = \frac{C}{V}\,|T_k|\,\hat{\mathbf{n}}$$

  where A_k is the area of face k, $\hat{\mathbf{n}}$ is the unit outward surface normal of face k, V is the volume enclosed by the surface, and $C = nRT$ is a constant that determines the original pressure.

The pressure depends on the volume of the object. As the object compresses, V becomes smaller and the pressure will rise. Computing the volume of an arbitrary shape seems like a daunting task, let's take a closer look

$$V = \iiint_\Omega 1 \, \mathrm{d}V$$

which we can write as

$$= \iiint_\Omega \nabla \cdot (0, 0, z) \, \mathrm{d}V$$

It's a divergence integral!

$$= \oiint_{\partial\Omega} (0, 0, z) \cdot \hat{\mathbf{n}} \, \mathrm{d}A$$

Our surface is made of faces, T_k, where each face has a constant unit outward normal,

$$= \sum_k \hat{n}_z \iint_{T_k} z \, \mathrm{d}A$$

Evaluating the triangle integral yields

$$= \sum_k \hat{n}_z \, |T_k| \, \frac{z_1 + z_2 + z_3}{3} = \sum_k \frac{n_z}{2} \frac{z_1 + z_2 + z_3}{3}$$

where the sum is taken over the triangular faces, $\hat{n}_z$ is the z-component of the unit outward face normal of T_k, and z_1, z_2, and z_3 are the z-components of the three nodes making up triangle T_k. Awesome!

You can generate spherical (or toroidal) triangular surface meshes of any size with the Thomson Problem Applet here: http://thomson.phy.syr.edu/thomsonapplet.php. Use the database or the algorithms to generate a good configuration and `File -> Point Set` and `File -> Triangle List` to save the point locations and triangle triplets.

Prompt 2 – Tetrahedral Mesh

Where Problem 1 attempts to use the mass-spring model on 2D surface meshes, we can also use a 3D tetrahedral mesh like those we saw in HW1. Tetrahedral meshes are in some ways more natural models for more solid objects (2D surface meshes are better for objects that consist of surfaces with internal pressure).

A tetrahedral mesh may have any features that we have discussed for the triangular `Mesh` in HW3 and the `Graph` in HW1 and HW2, but here are some operations you may want to consider implementing.

- First, we used viscous damping in HW2

$$\mathbf{f}_{i,\text{drag}} = -c\mathbf{v}_i$$

which damps all motion and is a decent model for nodes in a static fluid like air. But for 3D objects, this would also damp out rotations quite quickly as well. Instead of the spring force and viscous damping, try using a dashpot spring:

$$\mathbf{f}_{i,\text{dashpot}} = \sum_{j \in A_i} -\left(K_{ij}\left(|\mathbf{x}_i - \mathbf{x}_j| - L_{ij}\right) + C\frac{(\mathbf{v}_i - \mathbf{v}_j)\cdot(\mathbf{x}_i - \mathbf{x}_j)}{|\mathbf{x}_i - \mathbf{x}_j|}\right)\frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|}$$

where K_{ij} and L_{ij} is the spring constant and spring rest length as in HW2 and C is some damping constant. This models a spring with a damper attached to it so that motion in the direction of the spring force is damped while other motions such as rotations, translations, and other rigid body mechanics will be better preserved.

Tetrahedral mass-spring systems will often end up tying themselves in knots. There are a number of things you can do to try to prevent this.

- Using a tetrahedral mesh to model a volumetric mass-spring problem, we can add the constraint that no tetrahedron may invert. To do this, let $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ be the four vertices of a tetrahedron. Then,

$$6V = (\mathbf{x}_1 - \mathbf{x}_0)\cdot((\mathbf{x}_2 - \mathbf{x}_0)\times(\mathbf{x}_3 - \mathbf{x}_0))$$

defines the volume V of the tetrahedron (which can be negative depending on the orientation). The constraint that no tetrahedron may invert is equivalent to ensuring that each V does not change sign throughout the simulation. There are a number of ways this can be accomplished.

  - Hard constraint: Like the constraint application in HW2, we can attempt to find a way to fix an inverted tetrahedron by simply resetting the nodes. However, the fix may violate other constraints or cause neighbor tetrahedra to invert. Small timesteps would be required to make sure each violation was caught in time. This strategy is not recommended.
  - Volume penalty force: A better approach might be to add a force which is proportional to the change in volume of a tetrahedron, much like the change in length of an edge. A simple approach might be

$$\mathbf{f}_{i,\text{volume}} = \sum_{k \in \mathbb{T}(i)} -K(V_k - V_k^0)\frac{\mathbf{x}_i - \mathbf{p}_k}{|\mathbf{x}_i - \mathbf{p}_k|}$$

where K is a constant, $\mathbb{T}(i)$ is the set of tetrahedra adjacent to node i, V_k and V_k^0 are the current and original volume of tetrahedron k, and

$$\mathbf{p}_k = \frac{m_0\mathbf{x}_0 + m_1\mathbf{x}_1 + m_2\mathbf{x}_2 + m_3\mathbf{x}_3}{m_1 + m_2 + m_3 + m_4}$$

is the current barycenter of tetrahedron k. This force acts like a spring from $\mathbf{x}_i$ to the center of mass of tetrahedron k which attempts to restore it to its original volume.

– Or you could come up with your own mechanism. For instance, you could integrate forces and constraints into a single step, so that no Node position ever changes in a way that inverts a tetrahedron. For example, when a volume gets too small, the forces could be modified so that the tetrahedron's new volume will always be greater than or equal to the small volume.

Prompt 3 – Shallow Water Extensions

Suppose we wanted to cruise a boat across our shallow water. The boat exerts a downward force on the fluid, forcing fluid away from the boat and into the "columns" of fluid around it. Note that with our shallow water height field model, we assume that the object is small and floating on the water. This is because the water is incapable of "breaking" over an object; submersibles would require a different approach (but see below!). Furthermore, we assume the contact region between the water an an object is (nearly) convex for similar reasons.

The shallow water equations that we implemented are really a special case of the Euler equations with the hydrostatic equation of state

$$p(z) = \rho g(h - z)$$

where p is a pressure, ρ is the fluid density, g is the gravitational acceleration, h is the height of the water, and z is the height from the surface floor. Note that the pressure is zero at the surface of the water, $z = h$. When an object is floating on the water however, the surface pressure is nonzero and the pressure is modified to

$$p(z) = \rho g(h - z) + F/A$$

where F is a force in the z-direction bearing down on an area A of fluid. This modifies the shallow water flux by making the replacement

$$\frac{1}{2}gh^2 \;\rightarrow\; \frac{1}{2}gh^2 + \frac{Fh}{\rho A}$$

Recall that in the derivation of the flux function in HW3, the $\frac{1}{2}gh^2$ is averaged over the two incident triangles. The same must be done here for the new term $\frac{Fh}{\rho A}$.

Another improvement to the shallow water model is to include *bathymetry* – the shape of the surface the water flows over. Suppose we describe the bottom of the tub or pond with the function $b(\mathbf{x})$, which yields the height of the floor at position $\mathbf{x}$. The bathymetry appears as a source term in the momentum conservation equation:

$$\frac{\partial}{\partial t}\mathbf{Q} + \nabla \cdot \mathbf{F}(\mathbf{Q}) = \mathbf{S} \qquad\qquad \mathbf{S} = \begin{pmatrix} 0 \\ -gh\frac{\partial}{\partial x}b \\ -gh\frac{\partial}{\partial y}b \end{pmatrix}$$

where the derivatives of b can be approximated over each control volume.

Modify the shallow water simulation and flux functor so that an arbitrary force and source may be included on each triangle.

Prompt 4 – Collision Detector

In the most general case, collision detection is very difficult. However, some simple operations can be implemented that may provide sufficient results for our purposes.

- Interior testing. Wellformed meshes (triangular surfaces as in Problem 1 or tetrahedral volumes as in Problem 2) all have a closed triangular surface mesh. A clever way to determine whether a point is inside an object of this type is to draw a ray from the point out to infinity in any direction and count how many of these surface triangles that line intersects. If this crossing number is even, the point is outside the surface. If the crossing number is odd the point is inside the surface. To approximate full collision detection, you could simply check whether any vertex of one mesh is inside the other; if not, assume no collision.
- Culling and Morton codes. As in HW2, we can avoid the ray-triangle intersection test for the great majority of points at any time by keeping track of which objects are in which neighborhoods.

Prompt 5 – Visualizer Extensions

This is your chance to modify `SDLViewer`!

- **Interactivity.** It is practical to allow the `SDLViewer` to send out information about the mouse/keyboard events it receives from the user. We could then define forces on the events and make simulations interactive!

  Develop an extension to the viewer that is capable of registering "listener" objects which are notified whenever an event of a certain type is received. The "listener" can then use the event type and any additional information about the event (which key was pressed, the ray through the world that the mouse is currently over, etc). This can be accomplished in a number of ways:

  - Template the `SDLViewer` class on the type of a listener object. This listener object can then pass the `SDLViewer` a reference to itself and, provided it implements the correct functions, can be notified of `SDLEvents`.
  - Write an inner class in `SDLViewer` which uses virtual functions to specify the interface of a listener object. The listener object can then inherit from this inner class and pass the `SDLViewer` a pointer to itself.

A good test for this would be to add a new force or constraint to the `mass_spring` simulation that is modified by the user events. For example, a click could remove a node or edge, constrain a point, or apply a force to a set of points.

- **Vector field plotting.** In the dam-break simulation of HW3, you may have noticed a strong vortex form and remain stable for quite some time before dissipating. Although we could plot u, v, or $\sqrt{u^2 + v^2}$ in color using the current `SDLViewer`, it would also be useful to be able to plot a vector field. This could be used to plot the velocity vector of the fluid simulation or a mass-spring system. Design and implement an extension to the `SDLViewer` which allows a line representing a vector to be drawn from each point.

Prompt Zero – Optimization

Many of you have mentioned that your Meshes seem quite slow. Now's your chance to fix them. You may need to improve the runtime performance of your Meshes, in addition to another problem. A carefully measured and justified optimization pass could take the place of part or all of another problem, but discuss with an instructor first.

Prompt XX – Addition of your choosing

You may pitch your own problem to tackle. Some interesting ideas could be:

- Rigid body modeling.
- Optimization/Parallelization of your Graph and/or Mesh class.
- Automatic mesher. Take a set of point and return a Graph, Mesh or TetMesh. This would allow your Graph to change topology as the simulation continues.

Prompt YY – Research Projects

Alternatively, you may propose a unique project from your research or interests.

Please contact us by email with any projects plans related to Prompt YY by Wednesday, April 23rd.

Student Research

Propose a final project of your own, based on your own interests or research.

I recommend looking into libraries such as

- Thrust for GPU-acceleration.
- MLPACK for Scalable C++ Machine Learning.
- Boost.Python for integration into Python.

Cecka Research

In ongoing research, I have a few tree-related projects for up to two teams. This would involve building and integrating a number of interesting tree data structures into a generalized N-body library.

Contact me for more details.

Jones Research

As part of the Connectome project, Thouis (Ray) Jones is interested in a CS207 final project. He gives the following description of the project:

Input:

- A 3D volume with 64-bit integer labels
- A list of pairs of labels, indicating labels that have been merged, and should be treated as identical in all the operations below.

Operations we would like to support:

- Adjacency queries (which labels touch other labels, or touch the volume boundary)
- Bounding box of labels
- Bounding squares of labels (one or more 2D BBoxes per Z slice)
- (Others we don't necessarily know yet, such as bounding surface extraction)

The challenging part is that we want this to operate on multi-teravoxel volumes. Thus, it needs to be fast!

Contact me and Ray for more details, if interested.

Final Project Deliverables

In the course of the final project, your team is encouraged to work with and share codes with other teams. This can be especially useful in developing a complex system involving multiple prompts from above – extending the mesh, the viewer, the graph, optimizing pieces and algorithms, etc. For example, a Contact team (P4) and a Mesh-Spring team (P1) could simulate multiple, colliding inflated balls. A Shallow Water team (P3) and a Tetrahedron team (P2) could simulate a ball falling into the water. Of course, these are simple examples, and you are free to come up with your own demonstrations.

The deliverables of the final project are:

- A report describing the work accomplished, including a breakdown of individual contributions. We are happy to give feedback on drafts of this report, for instance to tell you if you've chosen too difficult a "novel computational project."

- Each team should provide specifications and documentation for their projects and what make clear which pieces they were responsible for. This specification is intended for your final project code clarity as well any any collaborating teams – so they can understand your design and integrate it with their own. We will also grade this specification. Specify in Doxygen comments; you may also include text files if you like.

- Each team should submit a working demonstration of their final project. With the code, submit a `README_FINAL.txt` that details the features of your project. This should include how we should run the code, any run-time commands or interaction, any interesting things we should try, etc.

- Each team should submit a review of any collaborations that were involved in the development of the final project in a `COLLAB_EVAL.txt` with the code. This should include detailing what resources the other team provided you, a code review of the code you received, and a general evaluation of the collaboration. What went right, what went wrong, what had to be changed, and what would you have done differently?

- Each student should individually submit, by email to the instructors, a group evaluation, `[NAME]_EVAL.txt`, that details your own contributions, those of your teammate, and your interaction with the other team.