Seismic Analysis Visualizer

CS 378H Final Project

Megan Mealey

Section 1

The goal of this project was to create a tool to visualize the output of an OpenSees simulation so that the user can better analyze the scenario being run. OpenSees is an open source structural analysis tool developed primarily by UC Berkeley. The core application is run through a command line interface. The user inputs information about nodes, beams, and other components of a structure as well as what forces are being applied and what to record, and OpenSees writes the results into a file. While text output can be useful in some cases, being able to visualize the structure and strain on each element can give better insight into how sound a structure is and what changes may need to be made. Additionally, WebGL allows the simulation to be viewed on different devices.

This project uses the openseespy library to run the opensees simulation. The example models used come from the opensees website and the openseespy website. The examples have three types of components: nodes, beams, and walls. The python program for each example runs the simulation and writes the data into a js file so that the WebGL program can read it.
Models can contain three types of components:

- Nodes - these are the joints in the model and are used to define endpoints for beams and walls. Nodes can be free or fixed. The recorders measure displacement of each node in the x, y, and z directions at each timestep.
- Beams - beams span between two nodes. Material parameters in the simulation define the strength of a beam and how it can move. The recorders measure the forces in the x, y, and z directions at each of the endpoints.
- Walls - walls can be defined by 4 nodes (a square) or 8 nodes (a prism). Like beams, walls have material parameters that determine how they move within the simulator. The recorders measure the net force on the wall in the x, y, and z directions.

The first key component of this project is taking large strings of data and creating an animated model. The parser reads the JSON string defining the structure and creates object arrays representing the nodes and elements, and these arrays are passed to the scene. The scene creates its own list of nodes stored as a list of starting positions and names, and elements are separated into “beam” and “wall” types and placed into arrays. Each beam and wall object contains references to the nodes that define it, and these are looked up by the shader at render time to determine the position of the object.

The forces on walls and beams are represented by how they are shaded. Each timestep is loaded in as a keyframe. Each keyframe has a time, a list of displacements for each node, and a list of forces on each object. At each timestep, the beams and walls within the model are shaded to show how much stress is being applied. The formula for the color is

`(stress_mag,0.6-stress_mag,0.6-stress_mag)`. For walls, stress_mag is the magnitude of the net force on the wall divided by the largest force on any wall in the simulation data. For beams, stress_mag is a similar ratio, but calculated for each endpoint of the beam and shaded between. The more red an object is, the closer it is to the maximum stress.

Section 2

The skinning project was used as starter code for this project, so the files are generally structured the same way. The source code can be compiled with make-seismic.py and the simulation can be viewed the same way the skinning project was.

The files for each example model are in src/seismic/static/static/assets, with each in a numbered folder. To collect the data and make it easy for javascript to read, I modified the python program for each example to copy the information into strings that are exported so that the main information loader can process them, similar to how the shader strings are formatted in the main program. The data for the selected model is loaded in by the SLoader class in StructParser.ts. SLoader. When the app needs to load a model in, it constructs a new SLoader with the model number as a parameter. The file paths and string names for each model are hard-coded in. A description of each model and link to the source can be found in the readme.

The mesh class was modified to create a mesh from nodes and elements (beams + walls). Beams and Walls each have a class to store object information, and the nodes are stored as a float array in the mesh. The nodes, beams, and walls of the mesh are rendered as points, lines, and trimeshes, respectively. Values like the maximum stress in the model are precomputed when the frames are loaded in and passed to the shader. The Gui file also sends information to the overlay.

While there are no major bugs I'm aware of, there are some limitations to this program. The first is that the examples and their file paths are hard-coded in. Additional work would need to be done to let the program read from an arbitrary file path. The data is also expected to be in a specific format and put into javascript strings so that the parser can read it. The program could probably be improved by being able to query the simulator instead of just read in pre-computed results. There are also many more possible queries that could be used and displayed, like moment and deformation.

Appendix - README

How to run:

In main folder, first run make-seismic.py, then run http-server dist -c-1 and view the model at 127.0.0.1:8080

To re-generate the example data, run the python program stored in an example folder found in src/seismic/static/static/assets

The controls for the WebGL window are the same as the skinning project with a few additions:

- Keys 1-9: Load the corresponding model
- Space: (During playback) toggle between play and pause
- T: toggle shadows
- -/+: (while paused) jump one frame forward/back
- Click on highlighted object: select object, info displayed in overlay
- Right click: clear selected object

Descriptions of examples

Gifs and videos of each example are in the artifacts folder

Example 1:
https://github.com/OpenSees/OpenSees/blob/master/EXAMPLES/ExamplePython/Example4.1.py
2D frame pushover analysis - a force is applied to the right side of the frame and increases until the target displacement is reached.

Example 2:
https://github.com/OpenSees/OpenSees/blob/master/EXAMPLES/ExamplePython/Example3.3.py
A simple frame in a simulated earthquake

Example 3:
https://github.com/OpenSees/OpenSees/blob/master/EXAMPLES/ExamplePython/Example5.1.py
A three-story frame in a simulated earthquake

Example 4: https://openseespydoc.readthedocs.io/en/latest/src/ThreeStorySteel.html
3-tier frame pushover analysis - a force is applied to the right side of the frame and increases until the target displacement is reached.

Example 5: https://openseespydoc.readthedocs.io/en/latest/src/nonlinearTruss.html
Nonlinear pushover analysis - the force on the top node increases linearly with time, and the displacement increases nonlinearly

Example 6: https://openseespydoc.readthedocs.io/en/latest/src/RCshearwall.html
Cyclic (back and forth) force is applied to the base of the wall

Example 7:
https://github.com/OpenSees/OpenSees/blob/master/EXAMPLES/ExamplePython/Example7.1.py
Vibrational force being applied to a 3D shell structure

Examples 8 and 9:
https://github.com/OpenSees/OpenSees/blob/master/EXAMPLES/ExamplePython/Example8.1.py
A large force is applied to a cantilever beam, and the beam vibrates as it returns to equilibrium