

# Lyra: A Thin Shell Simulator with FEM and FDM

You can know how to compile and use lyra in the documentation. For evaluating our method, please see `Report.pdf` and `Manuscript_Appendix.pdf` in root dictionary. For implementation, `include` fold include header and `src` include source code and `src/main.cpp` is entry for lyra. `offline_demo.mp4` is simple application for fun.

## Introduction

lyra is thin shell simulator name we give our program which solve PDE of deformable object by Finite Element Method (FEM) in space and Implicit Finite Difference Method (FDM) in time. Lyra provide large time step physical simulation by backward Euler. For evaluating internal force, specially, stretch force and shear force in plane and bending force out of plane, FEM is necessary numerical tool. Moreover, lyra provide simple external force implementation such as gravity, wind and point group constraint for generating interesting results. Unfortunately, robust thin shell simulator with collision detection and response is a hard and burdensome task, we don't implement those important and impressive traits in the simple simulator.

## Dependencies

- Eigen3
- Jsoncpp
- Alglib
- OpenGL (glfw3 glew)
- Stb\_image
- Boost (system filesystem)
- OpenPNG
- OpenMP\* (Optinal)

## Compile and Run

**ATTENTIONS:** you must use compiler supporting C++ 14, such as visual studio 2015 or g++ 3.5 above. For now, we successfully compile and run lyra in visual studio 2019 community and Ubuntu 18.04 LTS .

### MSVC

We code lyra with visual studio 2019 community , so compiling and run lyra in window is easy thing. we ommit concrete operator in the README.md.

### Linux

To compile lyra in linux, you need use apt-get to install dependencies below:

**ATTENTION:** If you encount 404 error when install dependencies, please use `sudo apt-get update` to update source and rerun commands. Source mirror is important, we succeeded to install all of dependencies in **Tsinghua Open Source Mirror**.

```
sudo apt-get update # optional

sudo apt-get install build-essential
sudo apt-get install cmake # version 3.0 above
sudo apt-get install libpng-dev
sudo apt-get install libboost-dev
sudo apt-get install libboost-filesystem-dev
sudo apt-get install libboost-serialization-dev
sudo apt-get install libeigen3-dev
```

```
sudo apt-get install libglew-dev
sudo apt-get install libglfw3-dev
```

and use cmake to configure in **root dictionary** of lyra:

```
cmake .
```

and then use make to compile lyra:

```
make -j16
```

When compile achieve, you can run program `./lyra conf/conf.json` in root dictionary. While lyra is running, you can adjust camera, Concrately,

- W **move camera forward**
- S **move camera back**
- A **move camera left**
- D **move camera right**
- ESC **exit lyra**
- Long press the left mouse button and move **adjust camera direction**

## Usage

---

If you successfully compile lyra, you can toy lyra with simulator configuration file `conf.json` in `./conf/` subdictionary by tuning parameters in the configuration file. Configuration file follow simple json format and provide some parameters to change. There is comment in detail for use `conf.json` :

```
{
  //time step for simulator
  "frame_time": 0.04,
  //sub time step for stable simulation
  "frame_steps": 8,
  // total running time
  "end_time": 15,
  //cloth info
  "cloths": [{
    //cloth mesh,ATTENTION:NOT SUPPORT ARBITRAY MESH,PLEASE USE MESH IN meshes subdictionary
    "mesh": "meshes/squarerefined.obj",
    //mesh transform
    "transform": {"translate": [-0.5, -1, 0.5]},
    //data from cloth material database of data-driven elastic models for cloth acquired by Huamin Wang,Ravi Ramamoorthi
    //you can see other avaiable material in ./materials/ fold.
    //damping is stiffness factor for damping force
    "materials": [{"data": "materials/gray-interlock.json",
      "damping": 0.03}],
    //texture is optinal and you can change any texture you like
    "texture":"textures/rainbow.jpg"
  }],
  //mesh handle
  "handles": [
    //pin points index in cloth
    {"nodes": [2,3],
      //motion is optinal for indicating handle motion
      "motion":0,
      //end_time is optinal for setting handle time
      "end_time":7}],
    //motion is optinal for setting motion trajectory
    "motions": [[
      //time specify start time of motion
      //transform specify transform
      {"time": 1, "transform": {"rotate": [0,0,0,1]}},
```

```
{ "time": 2, "transform": { "rotate": [-45,0,0,1] } },
{ "time": 2.5, "transform": { "rotate": [-90,0,0,1] } },
{ "time": 3, "transform": { "rotate": [-135,0,0,1] } },
{ "time": 3.5, "transform": { "rotate": [-180,0,0,1] } },
{ "time": 4, "transform": { "rotate": [-225,0,0,1] } },
{ "time": 4.5, "transform": { "rotate": [-270,0,0,1] } },
{ "time": 5, "transform": { "rotate": [-315,0,0,1] } },
{ "time": 6, "transform": { "rotate": [-360,0,0,1] } }
]],
//speed of gravity
"gravity": [0, 0, -9.8],
//wind is optional.
"wind":{
    //wind density
    "density":1,
    //wind velocity
    "velocity":[0,3,0],
    //wind drag
    "drag":1}
}
```

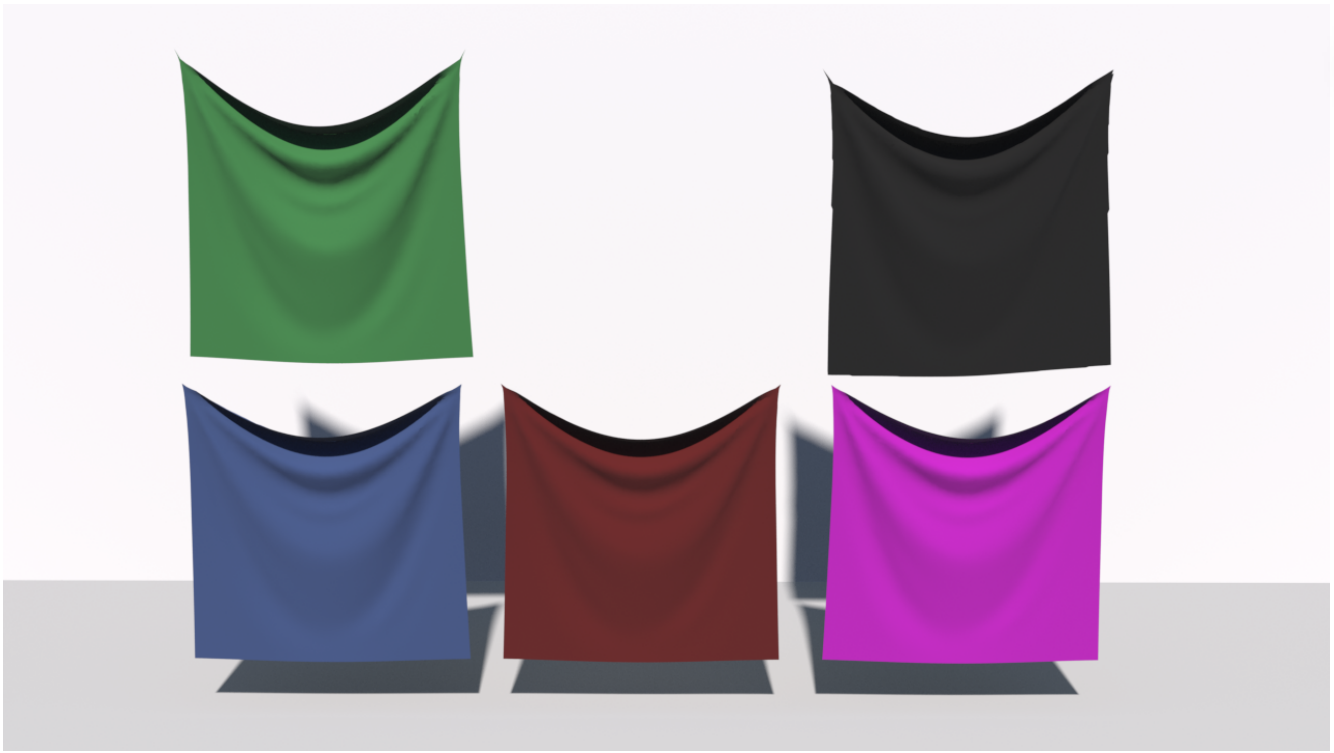
## Result Screenshot

PDE Final Project

- □ ×



Thin Shell Simulation Visualization Rendered by OpenGL(real-time)



Thin Shell Simulation Visualization Rendered by Houdini(offline)

## Code Hierarchy

---

Hierarchy of lyra is simple. The program is divided into two parts: `Render` which visualize simulation results and `Simulator` which simulate unstructured triangle mesh data by FEM and FDM. `Render` is simple OpenGL wrapper includes necessary visualized components: **camera**, **data buffer**, **window handle**, **shader** and **texture handle** etc. Some computer graphic knowledge is necessary for having a good grasp of `Render`.

Except packaging configuration file parser, `Simulator` implements simulation core of lyra in `physic.hpp` and `physic.cpp`, specially, `stretching_force` and `bending_force` functions, for every finite element in mesh, both of functions calculate small stiffness matrix and combine them into a large one. Next, in `ObtainImplicitEquation`, `Simulator` solve large linear system using wrapped linear system solver. This core loop implementation in `PhysicStep` which is a member function of `Simulator`.

## Acknowledgement

---

Although lyra is a simple thin shell simulator far too from particle application, It is not a easy task to implementing it from scratch using C++. we draw lessons from some excellent open source code such as [opencloth](#), [Argus](#) etc. For solving linear system, we reuse packaging eigen solver and fundamental data structure(vector, matrix and transformation etc) from [Argus](#). For thin shell motion parser we reuse motion parser from [opencloth](#).