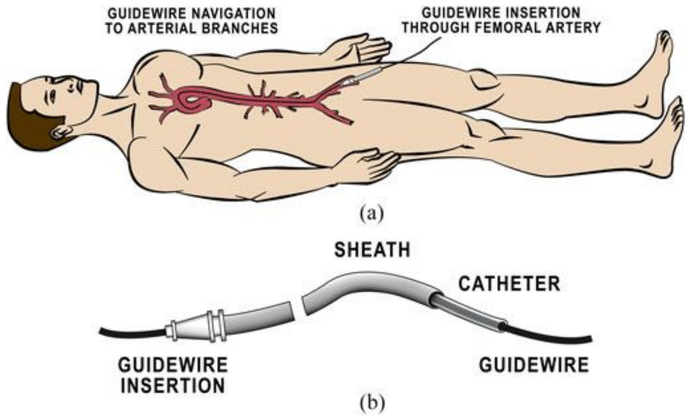
Improving Guidewire Simulations in Unity

Fortgeschrittenen Praktikum

Maximilian Maria Richter
Universität Heidelberg
Fakultät für Mathematik und Informatik
Betreuer: Prof. Jürgen Hesser, Lei Zheng

Guidewire Simulations

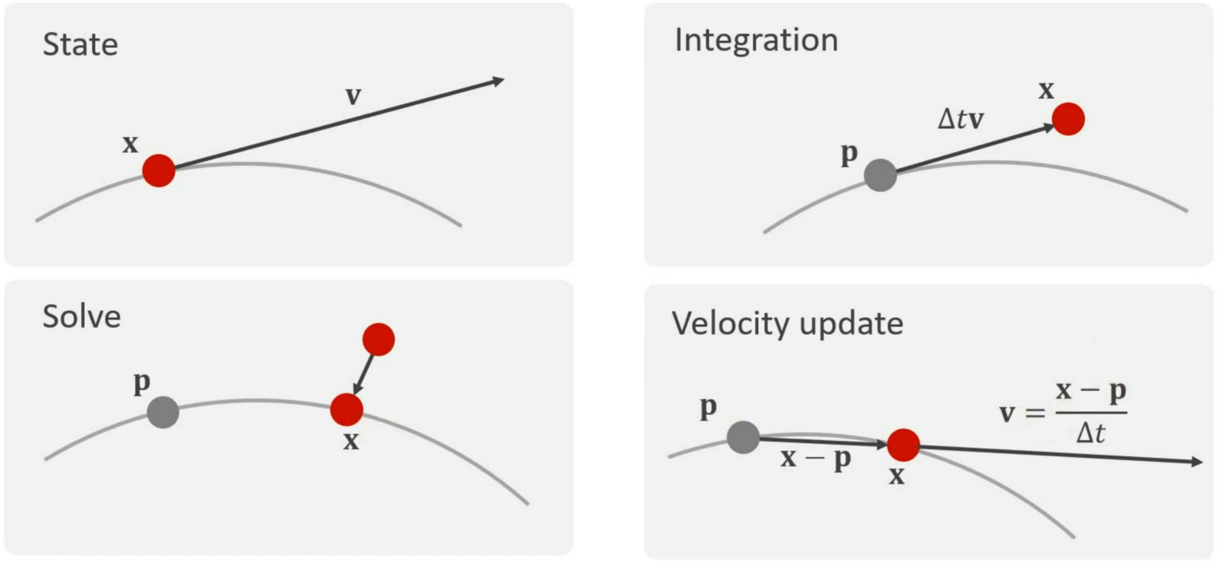
- Important for medicine
- Operations for cardiovascular diseases
- Difficult to simulate physics of guidewire


GUIDEWIRE NAVIGATION TO ARTERIAL BRANCHES
GUIDEWIRE INSERTION THROUGH FEMORAL ARTERY
(a)
SHEATH
CATHETER
GUIDEWIRE INSERTION
GUIDEWIRE
(b)

[1] https://www.researchgate.net/figure/Endovascular-procedure-basics-a-Insert-guidewire-into-femoral-artery-navigate-to_fig1_301719931

Position-Based Dynamics (PBD)

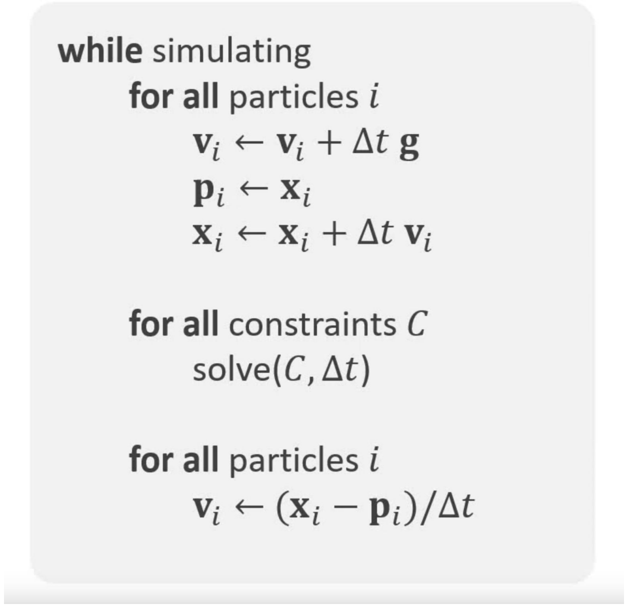
- Guidewire can be modeled as soft body
- Soft bodies can be simulated with PBD



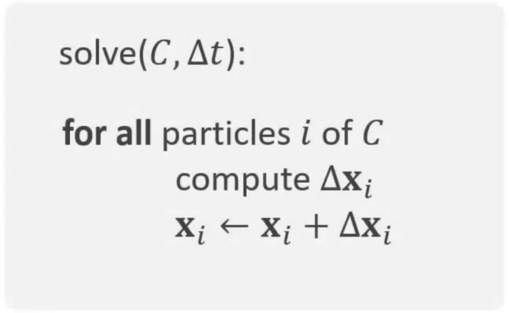
[1] https://www.youtube.com/watch?v=jrociOAYqxA

Position-Based Dynamics (PDB)

- Algorithm:



```
while simulating
    for all particles $i$
        $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t\ \mathbf{g}$
        $\mathbf{p}_i \leftarrow \mathbf{x}_i$
        $\mathbf{x}_i \leftarrow \mathbf{x}_i + \Delta t\ \mathbf{v}_i$

    for all constraints $C$
        solve($C, \Delta t$)

    for all particles $i$
        $\mathbf{v}_i \leftarrow (\mathbf{x}_i - \mathbf{p}_i)/\Delta t$
```

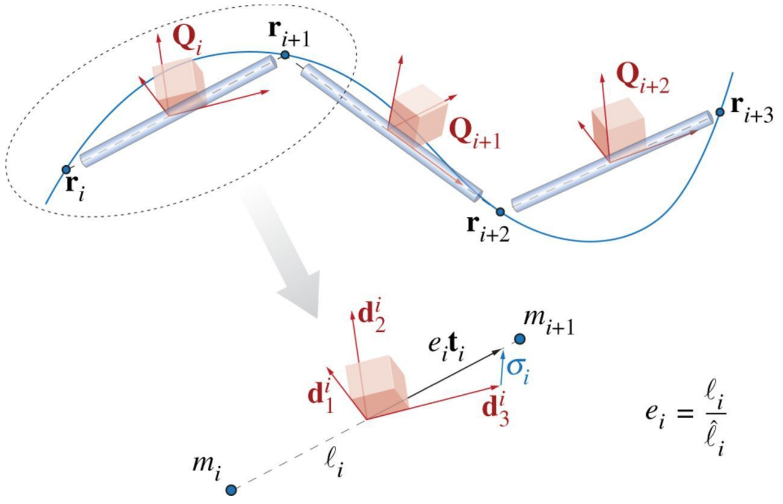


```
solve($C, \Delta t$):

for all particles $i$ of $C$
        compute $\Delta\mathbf{x}_i$
        $\mathbf{x}_i \leftarrow \mathbf{x}_i + \Delta\mathbf{x}_i$
```

[1] https://www.youtube.com/watch?v=jrociOAYqxA

Cosserat Rod Model

- Similar to Kirchhoff rods, but with twisting
- Position and Orientation-Based Dynamics (POBD) using quaternions
- Points (Spheres) and connecting rod elements with orientation



[1] https://www.cosseratrods.org/cosserat_rods/numerics/

Time Integration

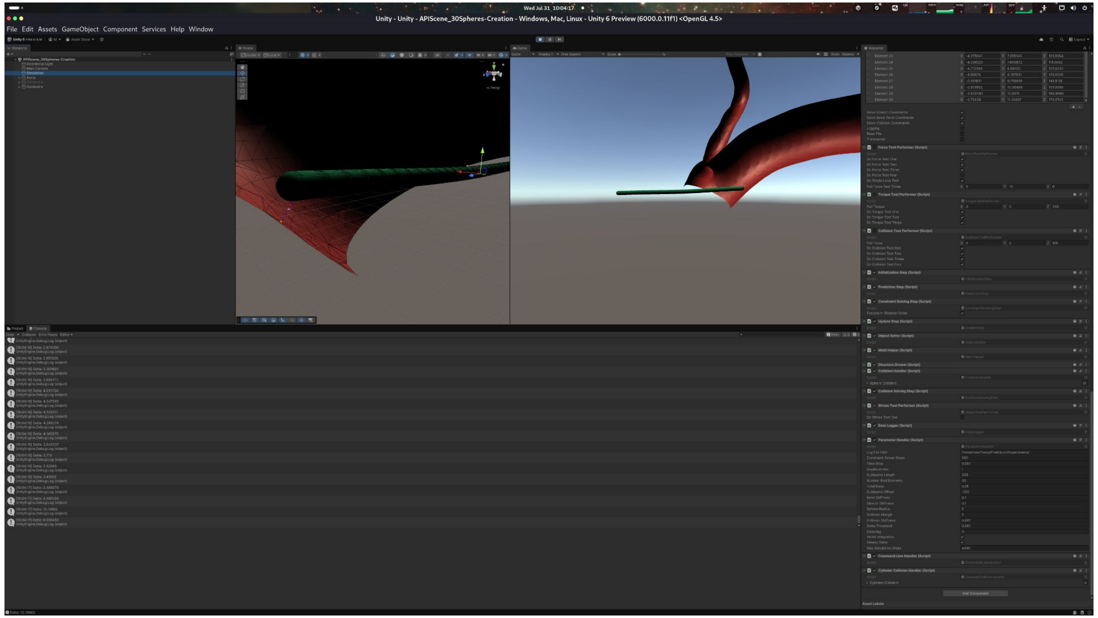
- Solve equations of motion numerically
- Explicit Euler Integration
- Simple but not very accurate

$$\mathbf{v}_{k+1} = \mathbf{v}_k + \Delta t \mathbf{a}_k,$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \mathbf{v}_k,$$

Implementation in Unity Game Engine

- Unity is a feature rich game engine using C#
- Guidewire simulation uses rendering and collision detection of unity


Unity - Unity - APIScene_30Spheres-Creation - Windows, Mac, Linux - Unity 6 Preview (6000.0.11f1) <OpenGL 4.5>
File Edit Assets GameObject Component Services Help Window

Collisions with Spheres

- Easy implementation, but rod element length limited

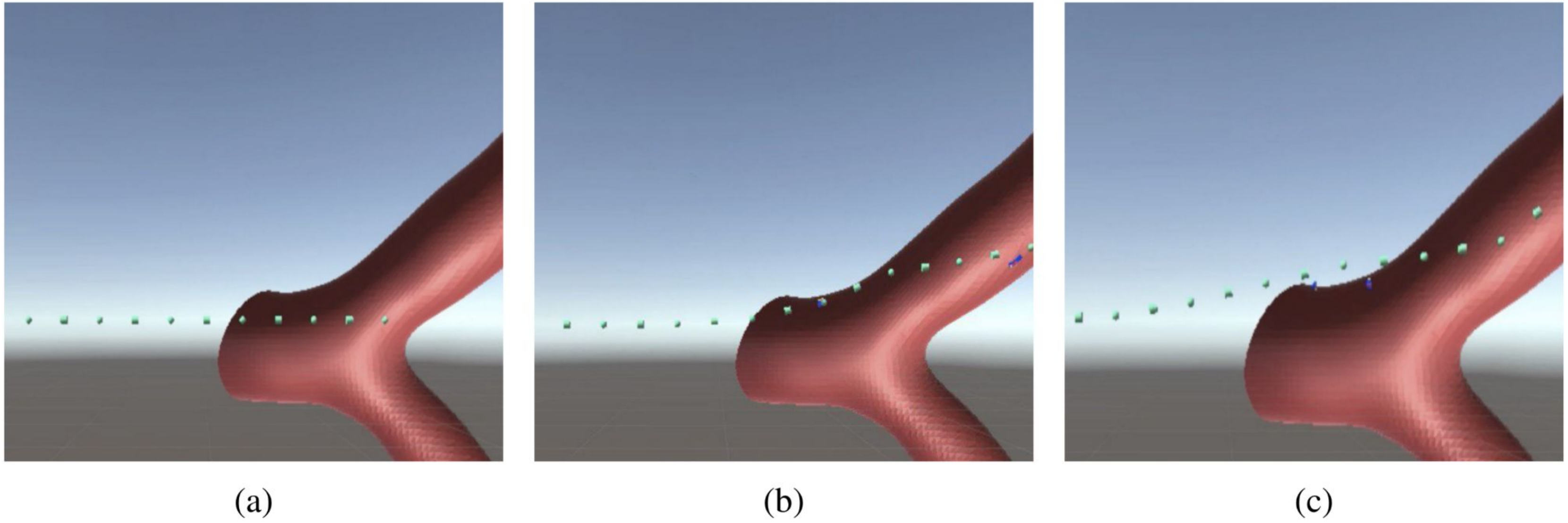


Figure 5.27: Illustration of the dynamics using a guidewire with large rodElementLength

[1] Convergence Analysis of Guidewire Simulations Employing Real Patient Data, A. Kreibich, 2023

Method

Simulation Improvements

Verlet Integration

- Verlet Scheme is symplectic second-order method
  - Conserves energy
- No need for velocities
  - Have to be computed explicitly if needed
- Update:

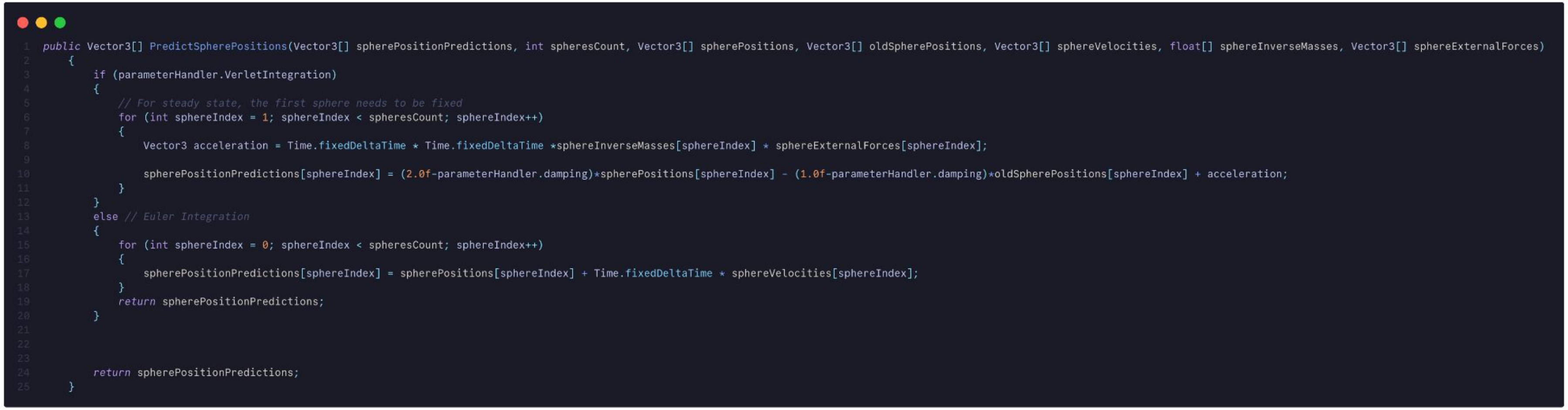
$$x_{n+1} = 2x_n - x_{n-1} + \Delta t^2 f(x_n)$$

- With damping

$$x_{n+1} = (2-\delta)x_n - (1-\delta)x_{n-1} + \Delta t^2 f(x_n)$$

- Use in prediction step

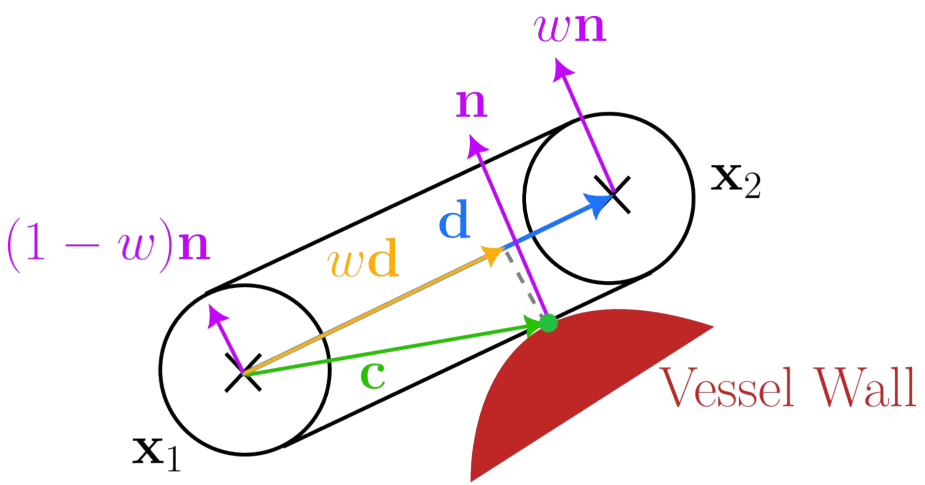
Verlet Integration



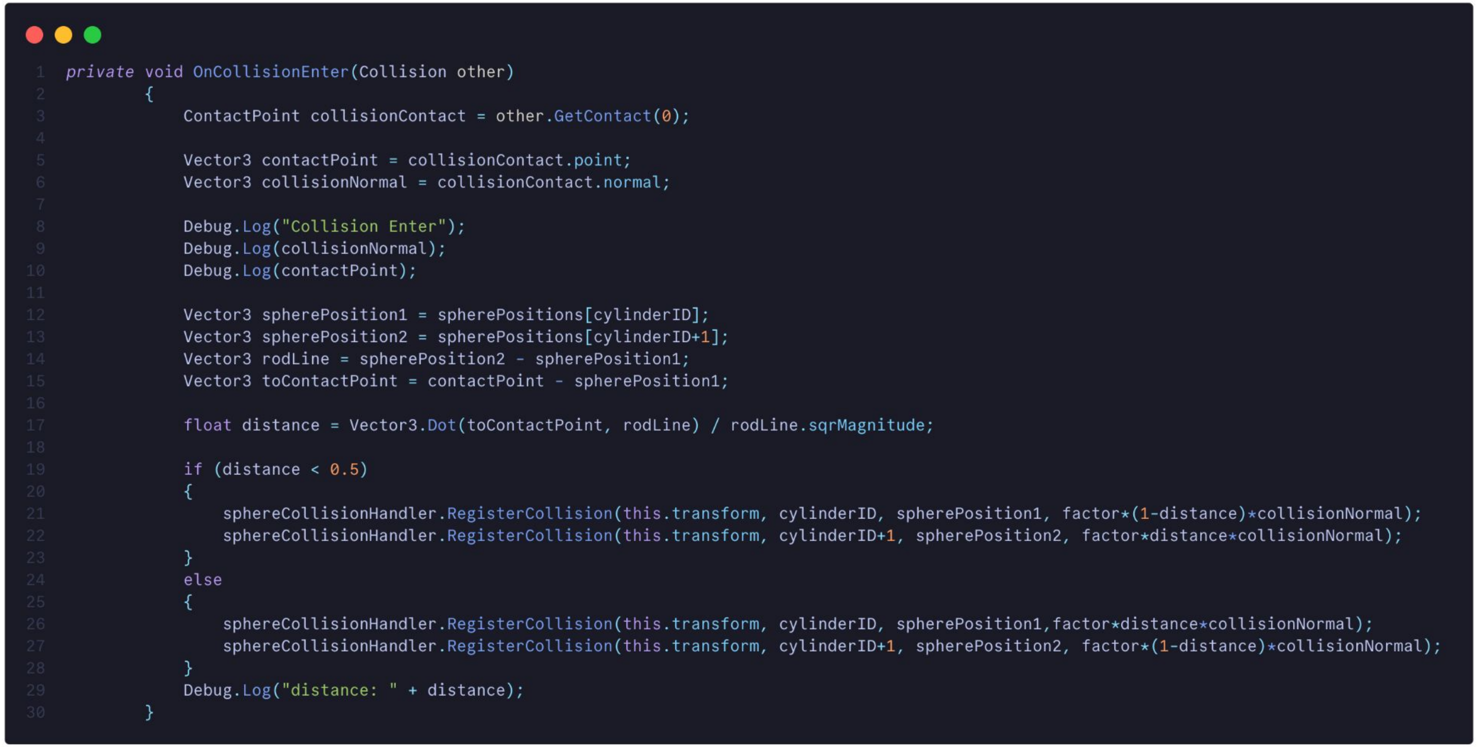
```
public Vector3[] PredictSpherePositions(Vector3[] spherePositionPredictions, int spheresCount, Vector3[] spherePositions, Vector3[] oldSpherePositions, Vector3[] sphereVelocities, float[] sphereInverseMasses, Vector3[] sphereExternalForces)
    {
        if (parameterHandler.VerletIntegration)
        {
            // For steady state, the first sphere needs to be fixed
            for (int sphereIndex = 1; sphereIndex < spheresCount; sphereIndex++)
            {
                Vector3 acceleration = Time.fixedDeltaTime * Time.fixedDeltaTime *sphereInverseMasses[sphereIndex] * sphereExternalForces[sphereIndex];

                spherePositionPredictions[sphereIndex] = (2.0f-parameterHandler.damping)*spherePositions[sphereIndex] - (1.0f-parameterHandler.damping)*oldSpherePositions[sphereIndex] + acceleration;
            }
        }
        else // Euler Integration
        {
            for (int sphereIndex = 0; sphereIndex < spheresCount; sphereIndex++)
            {
                spherePositionPredictions[sphereIndex] = spherePositions[sphereIndex] + Time.fixedDeltaTime * sphereVelocities[sphereIndex];
            }
            return spherePositionPredictions;
        }



        return spherePositionPredictions;
    }
```

Collisions with Cylinders

- Idea: Use collision constraint solving of spheres
- Distribute collision of cylinder onto neighboring spheres
- Weight the contribution to each sphere



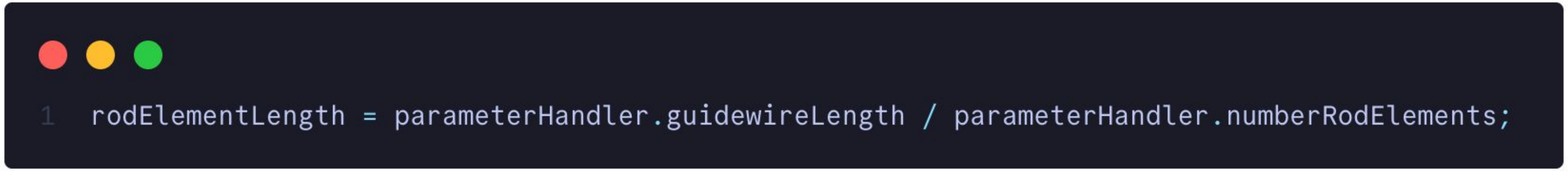
Collisions with Cylinders



```
private void OnCollisionEnter(Collision other)
        {
            ContactPoint collisionContact = other.GetContact(0);

            Vector3 contactPoint = collisionContact.point;
            Vector3 collisionNormal = collisionContact.normal;

            Debug.Log("Collision Enter");
            Debug.Log(collisionNormal);
            Debug.Log(contactPoint);

            Vector3 spherePosition1 = spherePositions[cylinderID];
            Vector3 spherePosition2 = spherePositions[cylinderID+1];
            Vector3 rodLine = spherePosition2 - spherePosition1;
            Vector3 toContactPoint = contactPoint - spherePosition1;

            float distance = Vector3.Dot(toContactPoint, rodLine) / rodLine.sqrMagnitude;

            if (distance < 0.5)
            {
                sphereCollisionHandler.RegisterCollision(this.transform, cylinderID, spherePosition1, factor*(1-distance)*collisionNormal);
                sphereCollisionHandler.RegisterCollision(this.transform, cylinderID+1, spherePosition2, factor*distance*collisionNormal);
            }
            else
            {
                sphereCollisionHandler.RegisterCollision(this.transform, cylinderID, spherePosition1,factor*distance*collisionNormal);
                sphereCollisionHandler.RegisterCollision(this.transform, cylinderID+1, spherePosition2, factor*(1-distance)*collisionNormal);
            }
            Debug.Log("distance: " + distance);
        }
```

Variable Rod Element Length

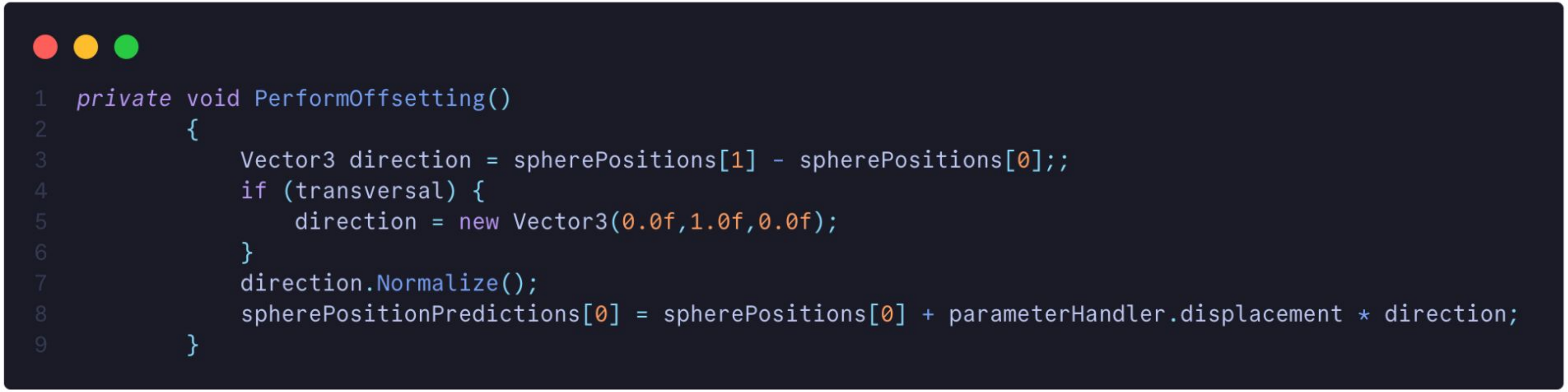
- Collision with cylinders enables rod elements longer than 2 * sphere radius



```
rodElementLength = parameterHandler.guidewireLength / parameterHandler.numberRodElements;
```

Control of Guidewire

- Instead of only displacement in one-direction
- Use first rod element as orientation for push



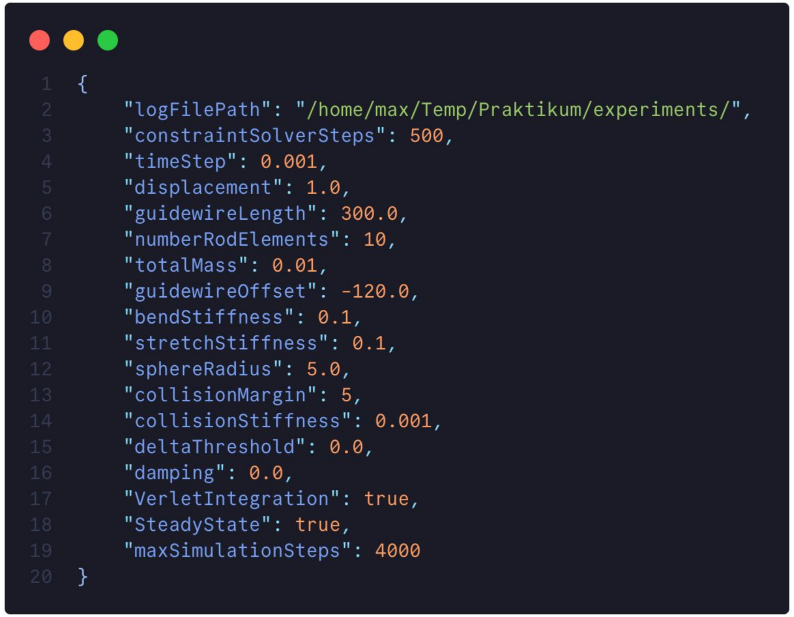
```
private void PerformOffsetting()
        {
            Vector3 direction = spherePositions[1] - spherePositions[0];;
            if (transversal) {
                direction = new Vector3(0.0f,1.0f,0.0f);
            }
            direction.Normalize();
            spherePositionPredictions[0] = spherePositions[0] + parameterHandler.displacement * direction;
        }
```

Method

Software Quality Improvements

JSON as Parameter Files

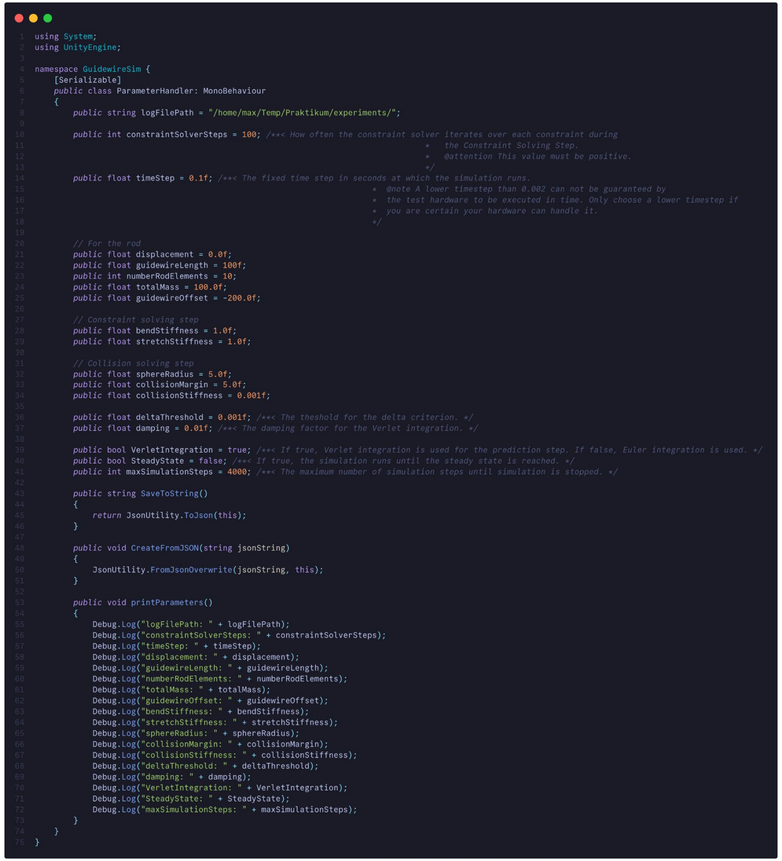
- JSON: Hierarchical data structure
- Change variables without need of recompiling the whole simulation
- Reproducibility of experiments



```json
{
    "logFilePath": "/home/max/Temp/Praktikum/experiments/",
    "constraintSolverSteps": 500,
    "timeStep": 0.001,
    "displacement": 1.0,
    "guidewireLength": 300.0,
    "numberRodElements": 10,
    "totalMass": 0.01,
    "guidewireOffset": -120.0,
    "bendStiffness": 0.1,
    "stretchStiffness": 0.1,
    "sphereRadius": 5.0,
    "collisionMargin": 5,
    "collisionStiffness": 0.001,
    "deltaThreshold": 0.0,
    "damping": 0.0,
    "VerletIntegration": true,
    "SteadyState": true,
    "maxSimulationSteps": 4000
}
```

Parameter Handler Class

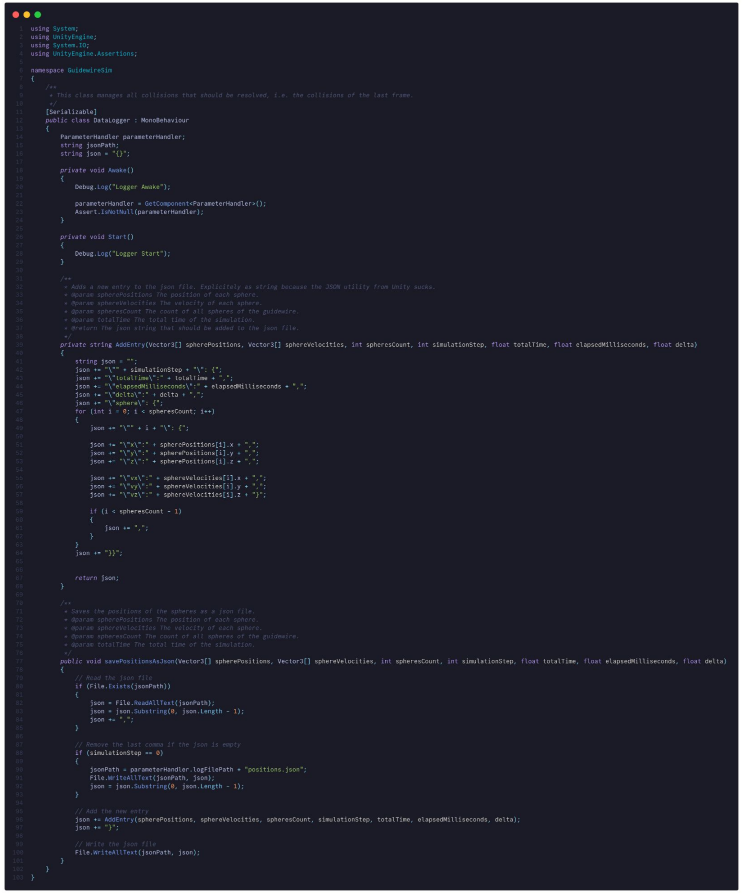
- Set and get all relevant parameters for the simulation
- Save and read JSON files



```csharp
using System;
using UnityEngine;

namespace GuidewireSim {
    [Serializable]
    public class ParameterHandler: MonoBehaviour
    {
        public string logFilePath = "/home/max/Temp/Praktikum/experiments/";

        public int constraintSolverSteps = 100; /**< How often the constraint solver iterates over each constraint during
                                                                                 *   the Constraint Solving Step.
                                                                                 *   @attention This value must be positive.
                                                                                 */
        public float timeStep = 0.1f; /**< The fixed time step in seconds at which the simulation runs.
                                                                      *  @note A lower timestep than 0.002 can not be guaranteed by
                                                                      *  the test hardware to be executed in time. Only choose a lower timestep if
                                                                      *  you are certain your hardware can handle it.
                                                                      */

        // For the rod
        public float displacement = 0.0f;
        public float guidewireLength = 100f;
        public int numberRodElements = 10;
        public float totalMass = 100.0f;
        public float guidewireOffset = -200.0f;

        // Constraint solving step
        public float bendStiffness = 1.0f;
        public float stretchStiffness = 1.0f;

        // Collision solving step
        public float sphereRadius = 5.0f;
        public float collisionMargin = 5.0f;
        public float collisionStiffness = 0.001f;

        public float deltaThreshold = 0.001f; /**< The theshold for the delta criterion. */
        public float damping = 0.01f; /**< The damping factor for the Verlet integration. */

        public bool VerletIntegration = true; /**< If true, Verlet integration is used for the prediction step. If false, Euler integration is used. */
        public bool SteadyState = false; /**< If true, the simulation runs until the steady state is reached. */
        public int maxSimulationSteps = 4000; /**< The maximum number of simulation steps until simulation is stopped. */

        public string SaveToString()
        {
            return JsonUtility.ToJson(this);
        }

        public void CreateFromJSON(string jsonString)
        {
            JsonUtility.FromJsonOverwrite(jsonString, this);
        }

        public void printParameters()
        {
            Debug.Log("logFilePath: " + logFilePath);
            Debug.Log("constraintSolverSteps: " + constraintSolverSteps);
            Debug.Log("timeStep: " + timeStep);
            Debug.Log("displacement: " + displacement);
            Debug.Log("guidewireLength: " + guidewireLength);
            Debug.Log("numberRodElements: " + numberRodElements);
            Debug.Log("totalMass: " + totalMass);
            Debug.Log("guidewireOffset: " + guidewireOffset);
            Debug.Log("bendStiffness: " + bendStiffness);
            Debug.Log("stretchStiffness: " + stretchStiffness);
            Debug.Log("sphereRadius: " + sphereRadius);
            Debug.Log("collisionMargin: " + collisionMargin);
            Debug.Log("collisionStiffness: " + collisionStiffness);
            Debug.Log("deltaThreshold: " + deltaThreshold);
            Debug.Log("damping: " + damping);
            Debug.Log("VerletIntegration: " + VerletIntegration);
            Debug.Log("SteadyState: " + SteadyState);
            Debug.Log("maxSimulationSteps: " + maxSimulationSteps);
        }
    }
}
```

Data Logger Class

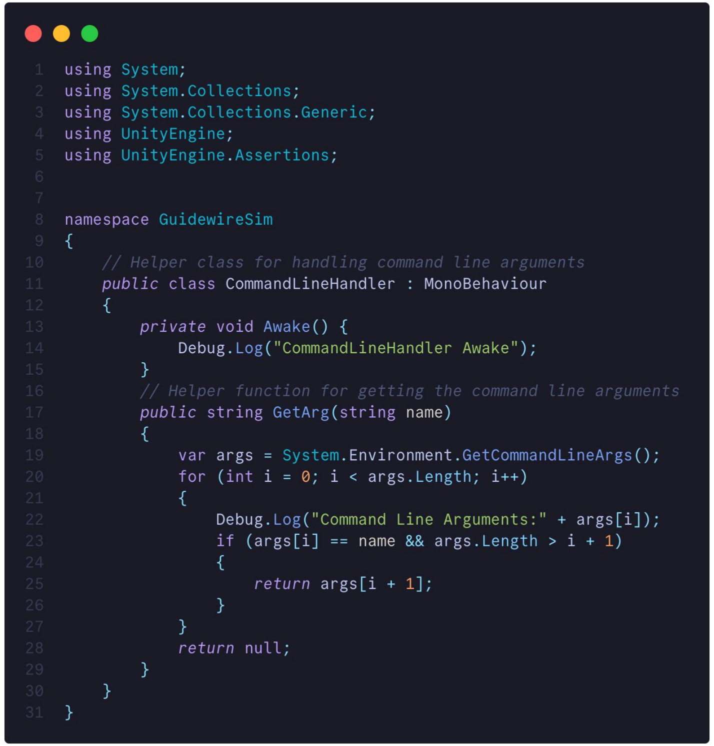
- Log the state of the system
- Export as JSON



```csharp
using System;
using UnityEngine;
using System.IO;
using UnityEngine.Assertions;

namespace GuidewireSim
{
    /**
     * This class manages all collisions that should be resolved, i.e. the collisions of the last frame.
     */
    [Serializable]
    public class DataLogger : MonoBehaviour
    {
        ParameterHandler parameterHandler;
        string jsonPath;
        string json = "{}";

        private void Awake()
        {
            Debug.Log("Logger Awake");

            parameterHandler = GetComponent<ParameterHandler>();
            Assert.IsNotNull(parameterHandler);
        }

        private void Start()
        {
            Debug.Log("Logger Start");
        }

        /**
         * Adds a new entry to the json file. Explicitely as string because the JSON utility from Unity sucks.
         * @param spherePositions The position of each sphere.
         * @param sphereVelocities The velocity of each sphere.
         * @param spheresCount The count of all spheres of the guidewire.
         * @param totalTime The total time of the simulation.
         * @return The json string that should be added to the json file.
         */
        private string AddEntry(Vector3[] spherePositions, Vector3[] sphereVelocities, int spheresCount, int simulationStep, float totalTime, float elapsedMilliseconds, float delta)
        {
            string json = "";
            json += "\"" + simulationStep + "\": {";
            json += "\"totalTime\":" + totalTime + ",";
            json += "\"elapsedMilliseconds\":" + elapsedMilliseconds + ",";
            json += "\"delta\":" + delta + ",";
            json += "\"sphere\": {";
            for (int i = 0; i < spheresCount; i++)
            {
                json += "\"" + i + "\": {";

                json += "\"x\":" + spherePositions[i].x + ",";
                json += "\"y\":" + spherePositions[i].y + ",";
                json += "\"z\":" + spherePositions[i].z + ",";

                json += "\"vx\":" + sphereVelocities[i].x + ",";
                json += "\"vy\":" + sphereVelocities[i].y + ",";
                json += "\"vz\":" + sphereVelocities[i].z + "}";

                if (i < spheresCount - 1)
                {
                    json += ",";
                }
            }
            json += "}}";


            return json;
        }

        /**
         * Saves the positions of the spheres as a json file.
         * @param spherePositions The position of each sphere.
         * @param sphereVelocities The velocity of each sphere.
         * @param spheresCount The count of all spheres of the guidewire.
         * @param totalTime The total time of the simulation.
         */
        public void savePositionsAsJson(Vector3[] spherePositions, Vector3[] sphereVelocities, int spheresCount, int simulationStep, float totalTime, float elapsedMilliseconds, float delta)
        {
            // Read the json file
            if (File.Exists(jsonPath))
            {
                json = File.ReadAllText(jsonPath);
                json = json.Substring(0, json.Length - 1);
                json += ",";
            }

            // Remove the last comma if the json is empty
            if (simulationStep == 0)
            {
                jsonPath = parameterHandler.logFilePath + "positions.json";
                File.WriteAllText(jsonPath, json);
                json = json.Substring(0, json.Length - 1);
            }

            // Add the new entry
            json += AddEntry(spherePositions, sphereVelocities, spheresCount, simulationStep, totalTime, elapsedMilliseconds, delta);
            json += "}";

            // Write the json file
            File.WriteAllText(jsonPath, json);
        }
    }
}
```

Command Line Handler Class

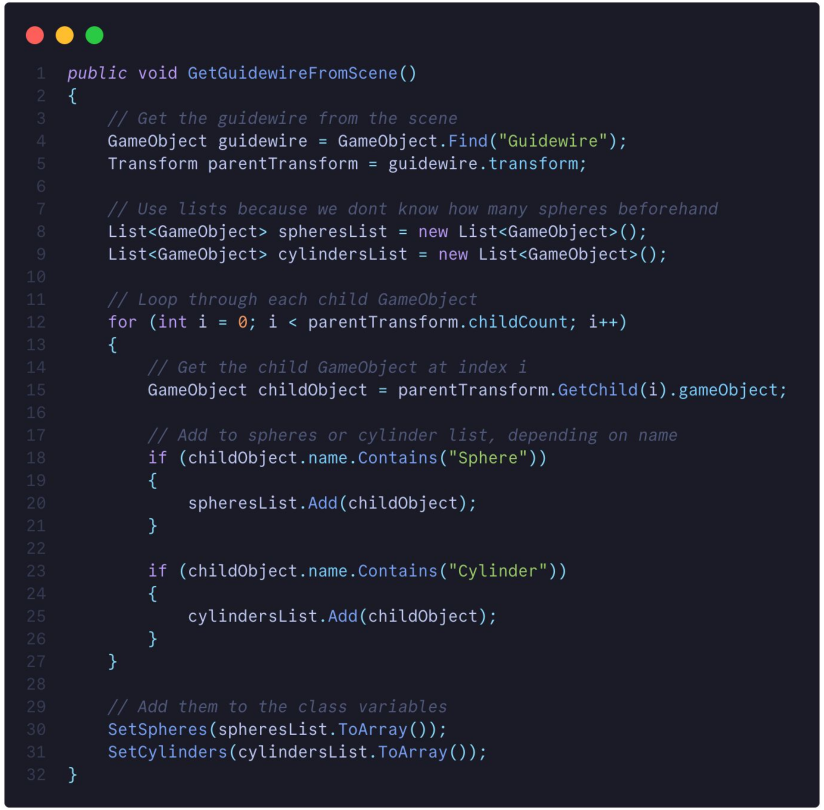
- Provide clean interface to command line
- Used to import
  - parameters.json file path
  - log.txt file path for unity log



```csharp
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Assertions;


namespace GuidewireSim
{
    // Helper class for handling command line arguments
    public class CommandLineHandler : MonoBehaviour
    {
        private void Awake() {
            Debug.Log("CommandLineHandler Awake");
        }
        // Helper function for getting the command line arguments
        public string GetArg(string name)
        {
            var args = System.Environment.GetCommandLineArgs();
            for (int i = 0; i < args.Length; i++)
            {
                Debug.Log("Command Line Arguments:" + args[i]);
                if (args[i] == name && args.Length > i + 1)
                {
                    return args[i + 1];
                }
            }
            return null;
        }
    }
}
```

Get Guidewire from Scene

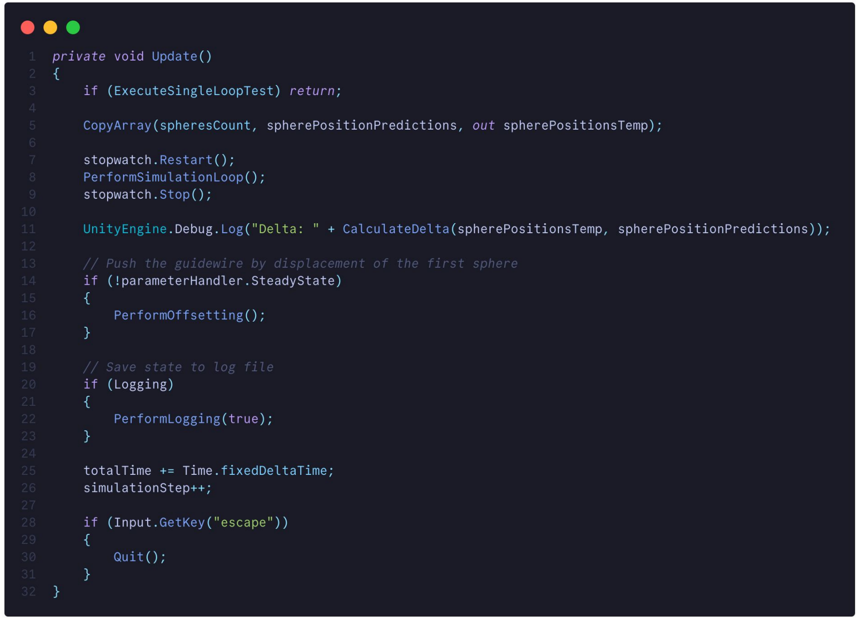
- If guidewire not created at runtime
- Assign guidewire correctly to simulation loop



```
public void GetGuidewireFromScene()
{
    // Get the guidewire from the scene
    GameObject guidewire = GameObject.Find("Guidewire");
    Transform parentTransform = guidewire.transform;

    // Use lists because we dont know how many spheres beforehand
    List<GameObject> spheresList = new List<GameObject>();
    List<GameObject> cylindersList = new List<GameObject>();

    // Loop through each child GameObject
    for (int i = 0; i < parentTransform.childCount; i++)
    {
        // Get the child GameObject at index i
        GameObject childObject = parentTransform.GetChild(i).gameObject;

        // Add to spheres or cylinder list, depending on name
        if (childObject.name.Contains("Sphere"))
        {
            spheresList.Add(childObject);
        }

        if (childObject.name.Contains("Cylinder"))
        {
            cylindersList.Add(childObject);
        }
    }

    // Add them to the class variables
    SetSpheres(spheresList.ToArray());
    SetCylinders(cylindersList.ToArray());
}
```

Update method

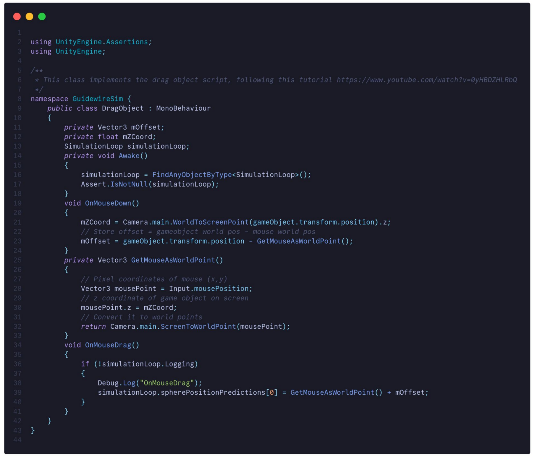
- Unity has two methods for simulations:
  - **Update**: called once per frame
  - **FixedUpdate**: called at fixed time interval
- Result: Time step is constant



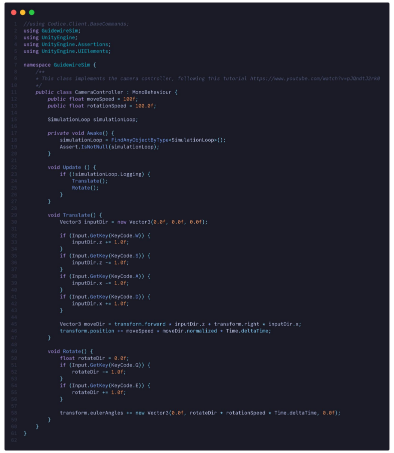
```csharp
private void Update()
{
    if (ExecuteSingleLoopTest) return;

    CopyArray(spheresCount, spherePositionPredictions, out spherePositionsTemp);

    stopwatch.Restart();
    PerformSimulationLoop();
    stopwatch.Stop();

    UnityEngine.Debug.Log("Delta: " + CalculateDelta(spherePositionsTemp, spherePositionPredictions));

    // Push the guidewire by displacement of the first sphere
    if (!parameterHandler.SteadyState)
    {
        PerformOffsetting();
    }

    // Save state to log file
    if (Logging)
    {
        PerformLogging(true);
    }

    totalTime += Time.fixedDeltaTime;
    simulationStep++;

    if (Input.GetKey("escape"))
    {
        Quit();
    }
}
```

Mouse and Keyboard Interaction

- To give an initial implementation for the external control of the guidewire
- Using unity engine peripheral control
- Mouse can pick first sphere and push guirewire through vessel



```
using UnityEngine.Assertions;
using UnityEngine;

/**
 * This class implements the drag object script, following this tutorial https://www.youtube.com/watch?v=0yHBDZHLRbQ
 */
namespace GuidewireSim {
    public class DragObject : MonoBehaviour
    {
        private Vector3 mOffset;
        private float mZCoord;
        SimulationLoop simulationLoop;
        private void Awake()
        {
            simulationLoop = FindAnyObjectByType<SimulationLoop>();
            Assert.IsNotNull(simulationLoop);
        }
        void OnMouseDown()
        {
            mZCoord = Camera.main.WorldToScreenPoint(gameObject.transform.position).z;
            // Store offset = gameobject world pos - mouse world pos
            mOffset = gameObject.transform.position - GetMouseAsWorldPoint();
        }
        private Vector3 GetMouseAsWorldPoint()
        {
            // Pixel coordinates of mouse (x,y)
            Vector3 mousePoint = Input.mousePosition;
            // z coordinate of game object on screen
            mousePoint.z = mZCoord;
            // Convert it to world points
            return Camera.main.ScreenToWorldPoint(mousePoint);
        }
        void OnMouseDrag()
        {
            if (!simulationLoop.Logging)
            {
                Debug.Log("OnMouseDrag");
                simulationLoop.spherePositionPredictions[0] = GetMouseAsWorldPoint() + mOffset;
            }
        }
    }
}
```



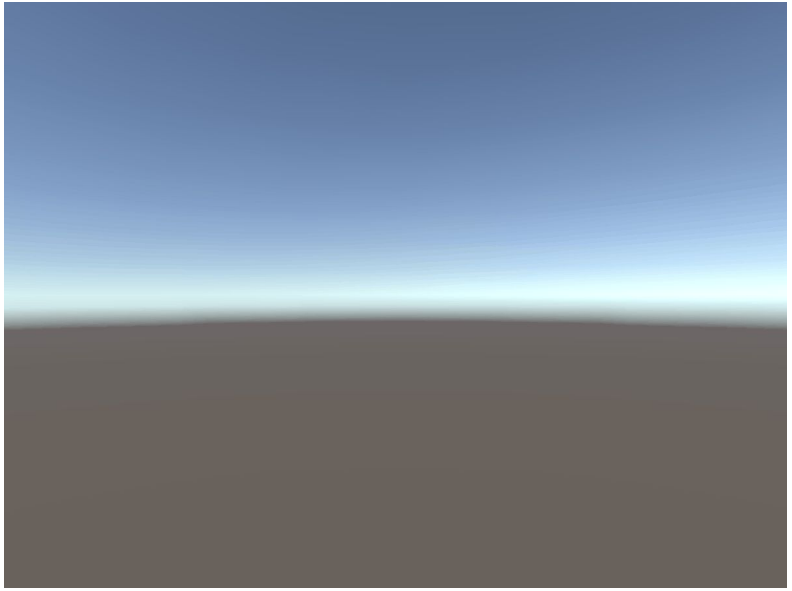
```
//using Codice.Client.BaseCommands;
using GuidewireSim;
using UnityEngine;
using UnityEngine.Assertions;
using UnityEngine.UIElements;

namespace GuidewireSim {
    /**
    * This class implements the camera controller, following this tutorial https://www.youtube.com/watch?v=pJQndtJ2rk0
    */
    public class CameraController : MonoBehaviour {
        public float moveSpeed = 100f;
        public float rotationSpeed = 100.0f;

        SimulationLoop simulationLoop;

        private void Awake() {
            simulationLoop = FindAnyObjectByType<SimulationLoop>();
            Assert.IsNotNull(simulationLoop);
        }

        void Update () {
            if (!simulationLoop.Logging) {
                Translate();
                Rotate();
            }
        }

        void Translate() {
            Vector3 inputDir = new Vector3(0.0f, 0.0f, 0.0f);

            if (Input.GetKey(KeyCode.W)) {
                inputDir.z += 1.0f;
            }
            if (Input.GetKey(KeyCode.S)) {
                inputDir.z -= 1.0f;
            }
            if (Input.GetKey(KeyCode.A)) {
                inputDir.x -= 1.0f;
            }
            if (Input.GetKey(KeyCode.D)) {
                inputDir.x += 1.0f;
            }

            Vector3 moveDir = transform.forward * inputDir.z + transform.right * inputDir.x;
            transform.position += moveSpeed * moveDir.normalized * Time.deltaTime;
        }

        void Rotate() {
            float rotateDir = 0.0f;
            if (Input.GetKey(KeyCode.Q)) {
                rotateDir -= 1.0f;
            }
            if (Input.GetKey(KeyCode.E)) {
                rotateDir += 1.0f;
            }

            transform.eulerAngles += new Vector3(0.0f, rotateDir * rotationSpeed * Time.deltaTime, 0.0f);
        }
    }
}
```

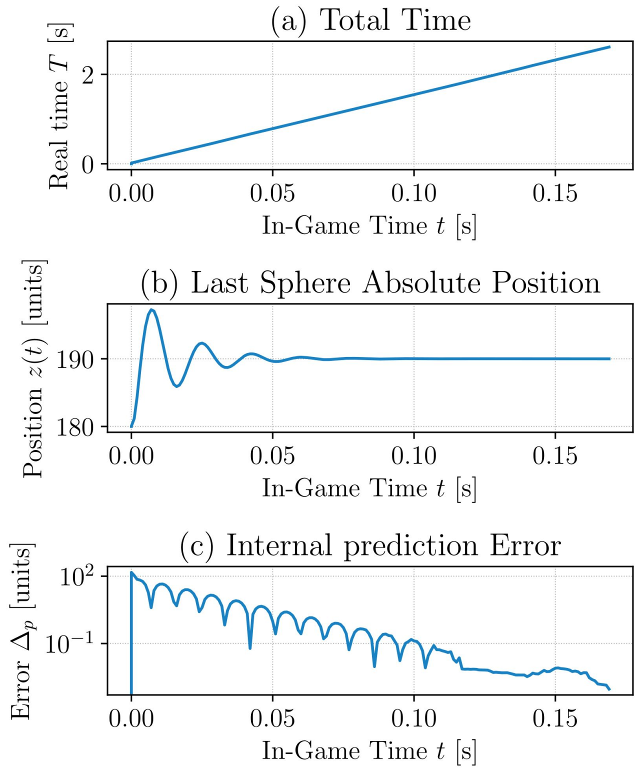
Results

Convergence Study

Longitudinal perturbation

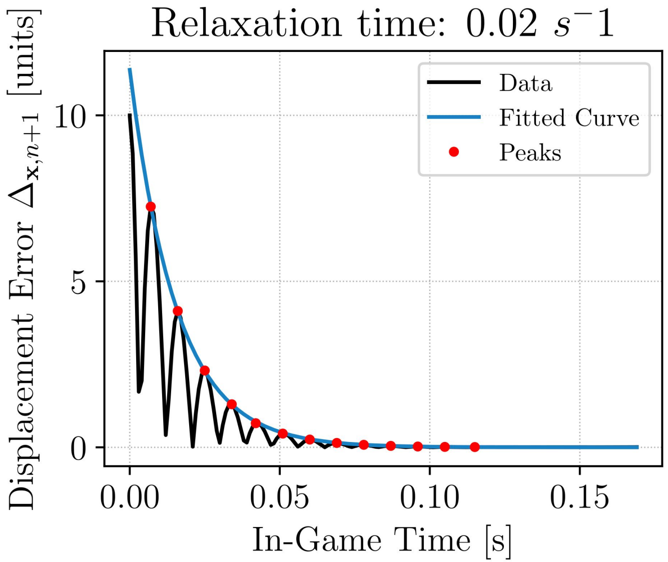
- Wait until convergence
- Measure decay time of damped oscillator

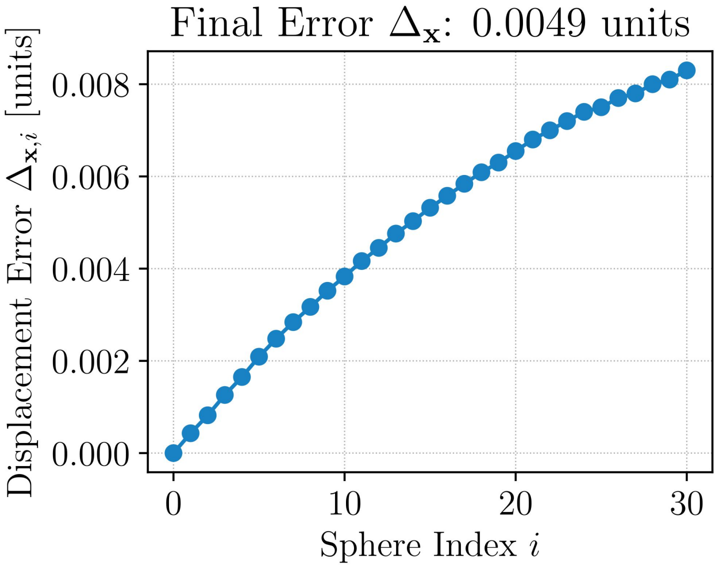




Longitudinal perturbation

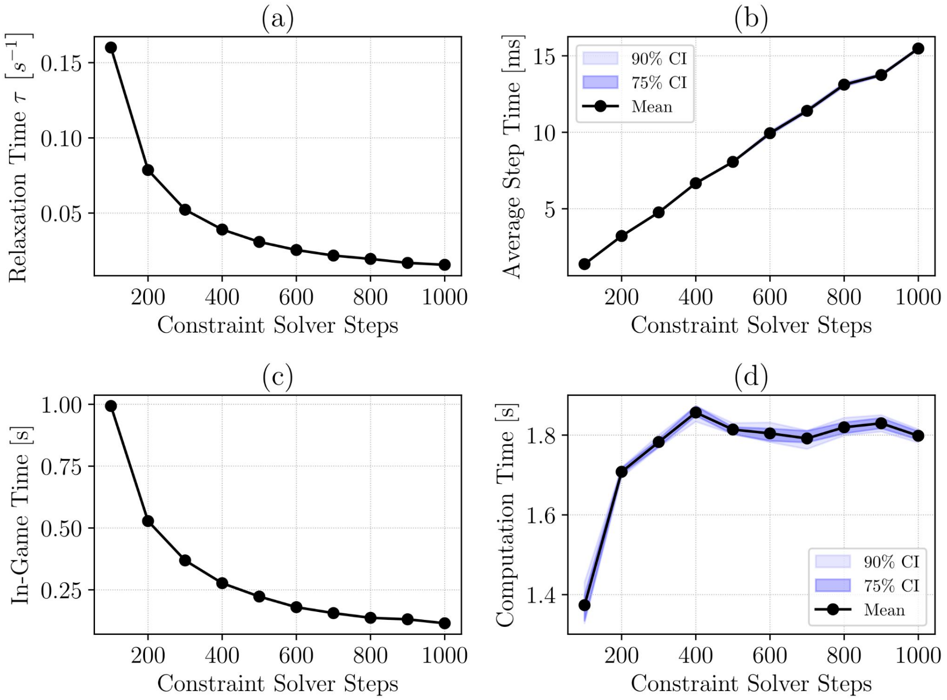
- Wait until convergence
- Measure decay time of damped oscillator





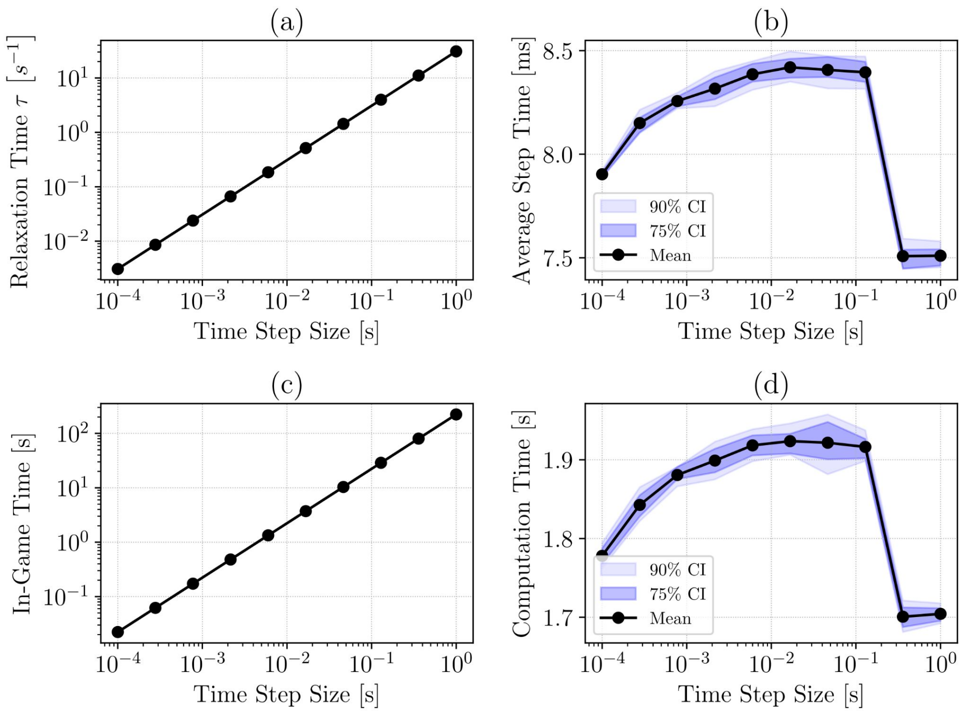
Longitudinal Perturbation

Constraint solver steps



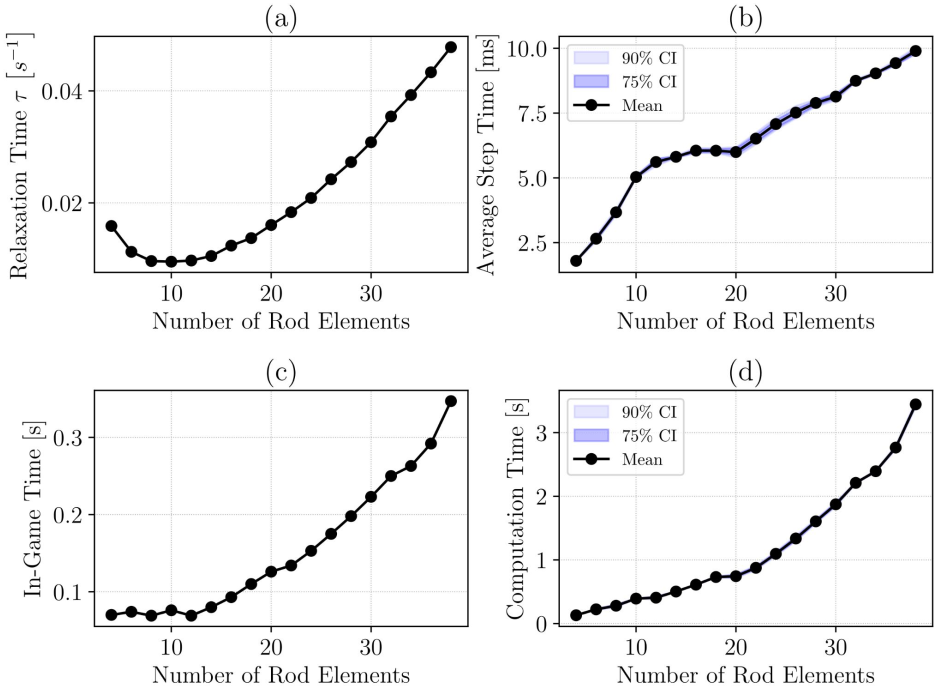
Longitudinal Perturbation

Time Step Size



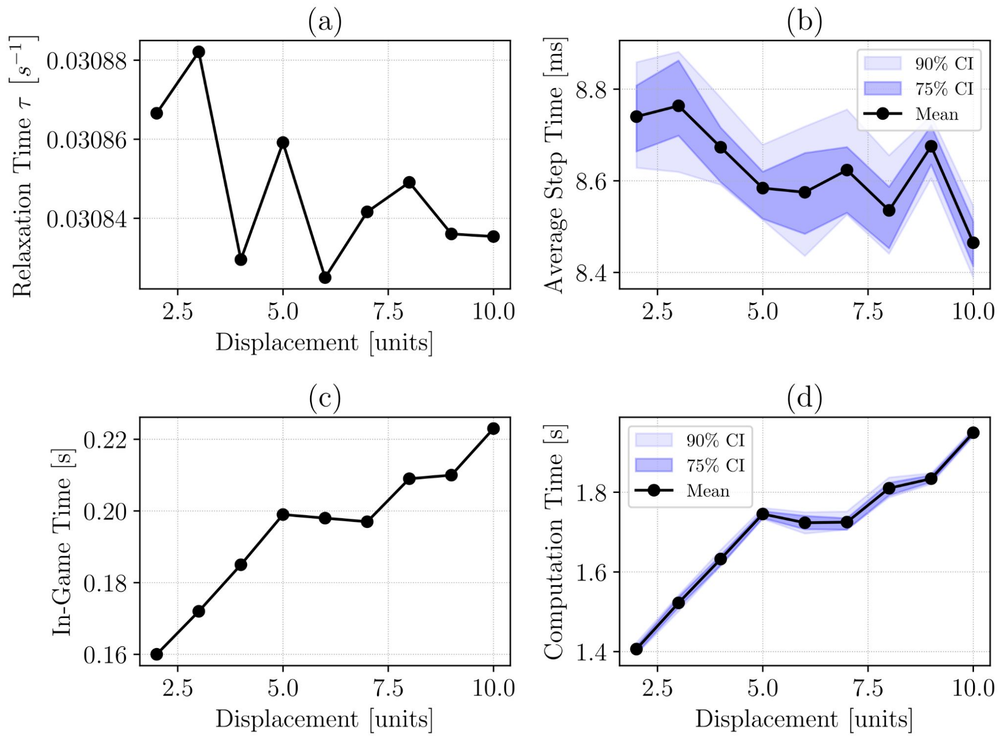
Longitudinal Perturbation

Number of Rod Elements



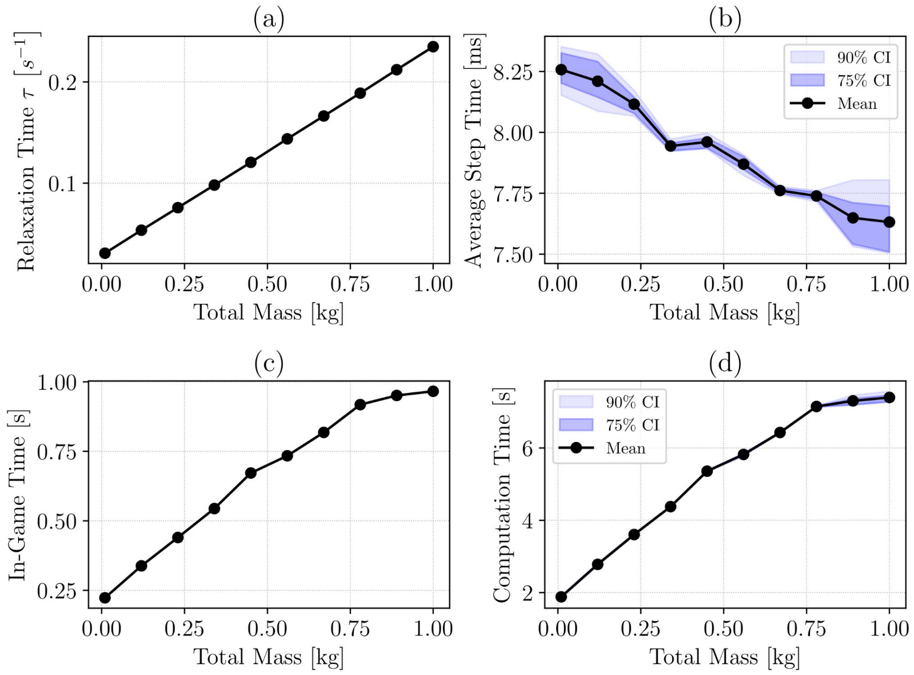
Longitudinal Perturbation

Displacement



Longitudinal Perturbation

Total Mass

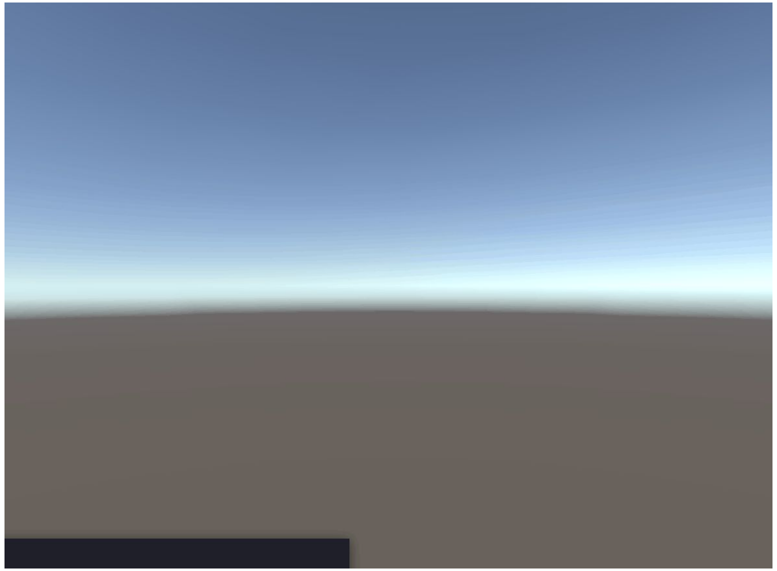


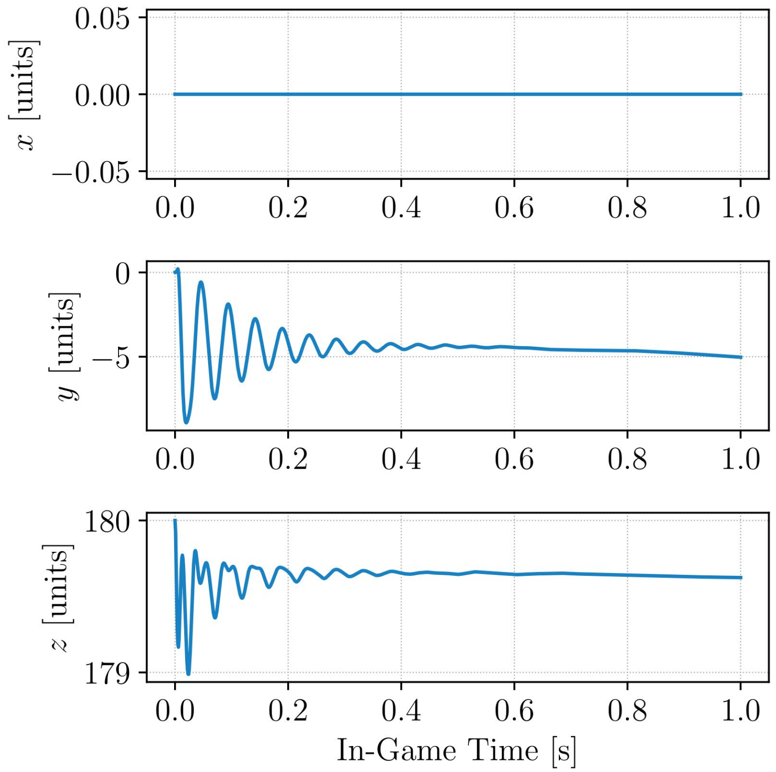
Results

Transversal Accuracy Study

Transversal Perturbation

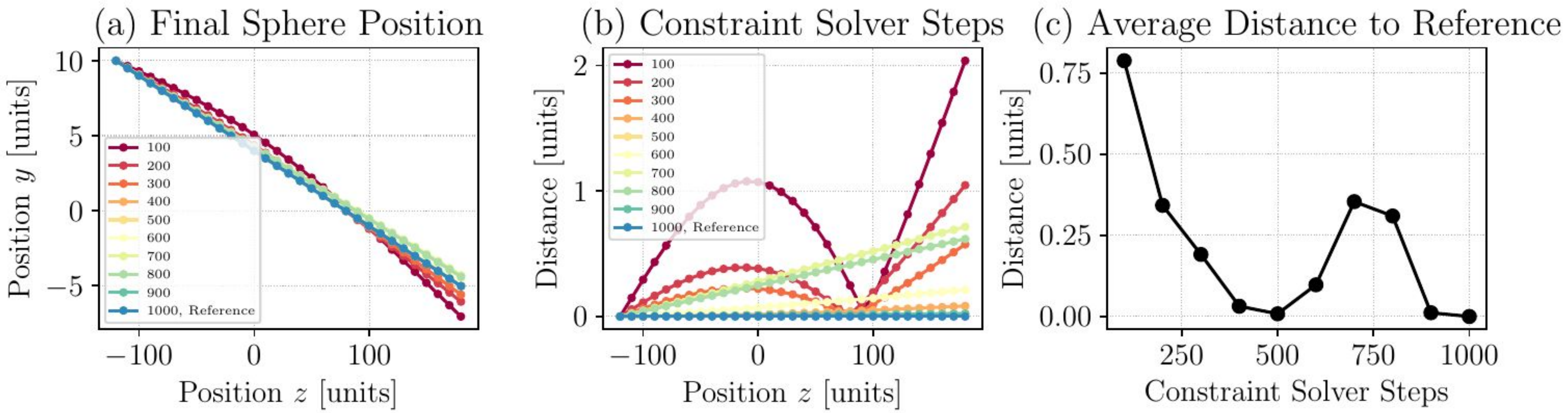
- Offset along y-axis
- Standing wave develops
- Reference solution per parameter





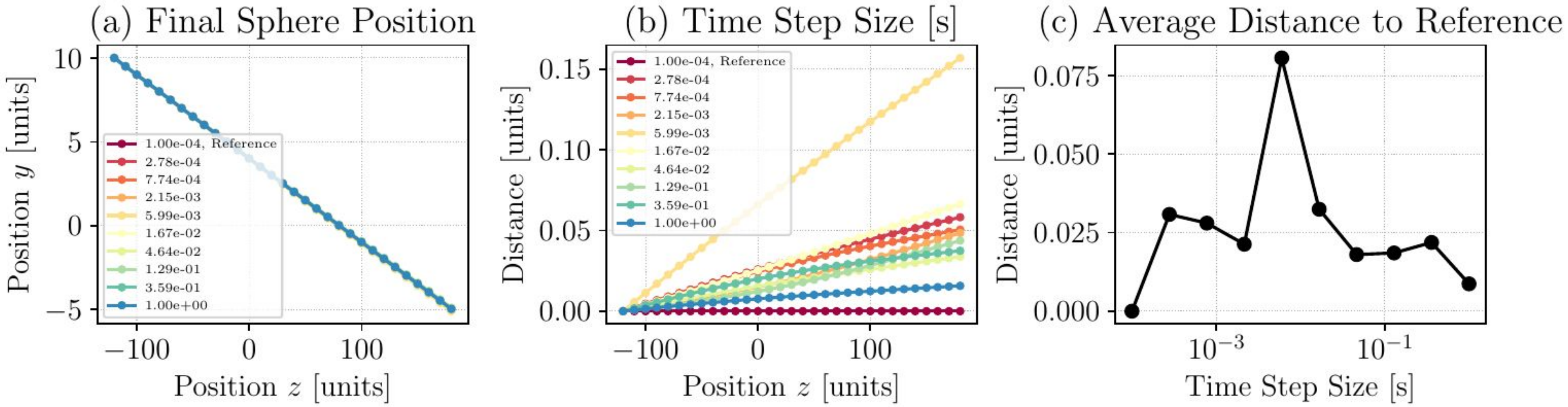
Transversal Perturbation

Constraint Solver Steps



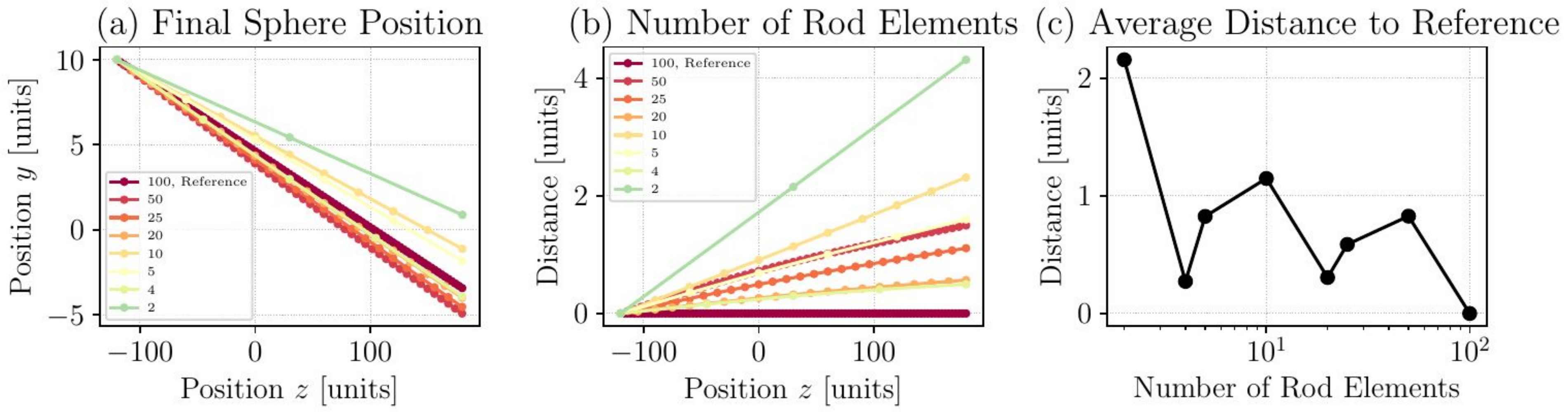
Transversal Perturbation

Time Steps



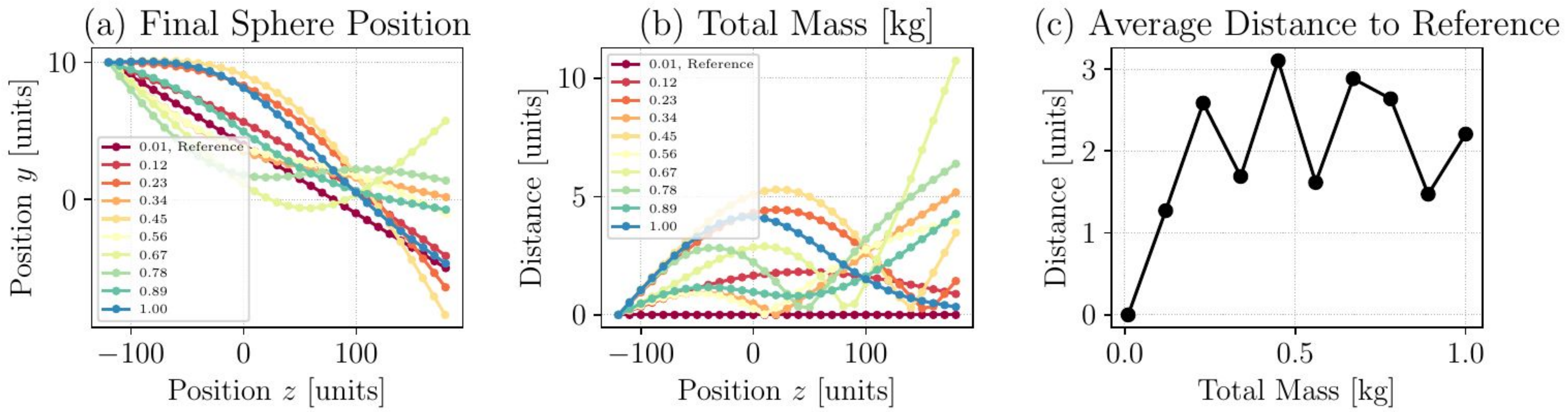
Transversal Perturbation

Number of Rod Elements



Transversal Perturbation

Total Mass

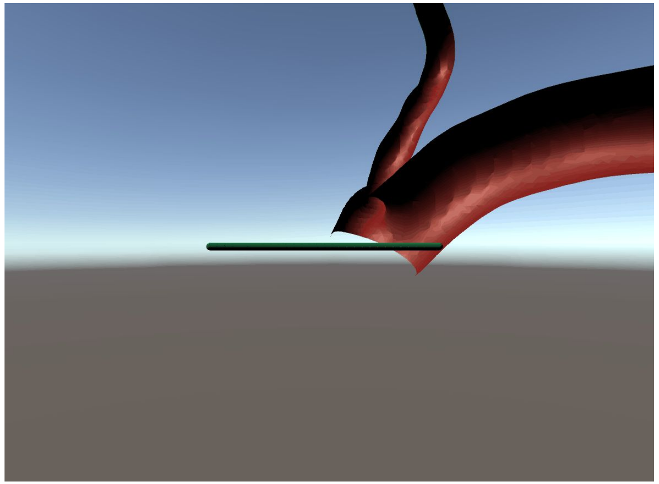


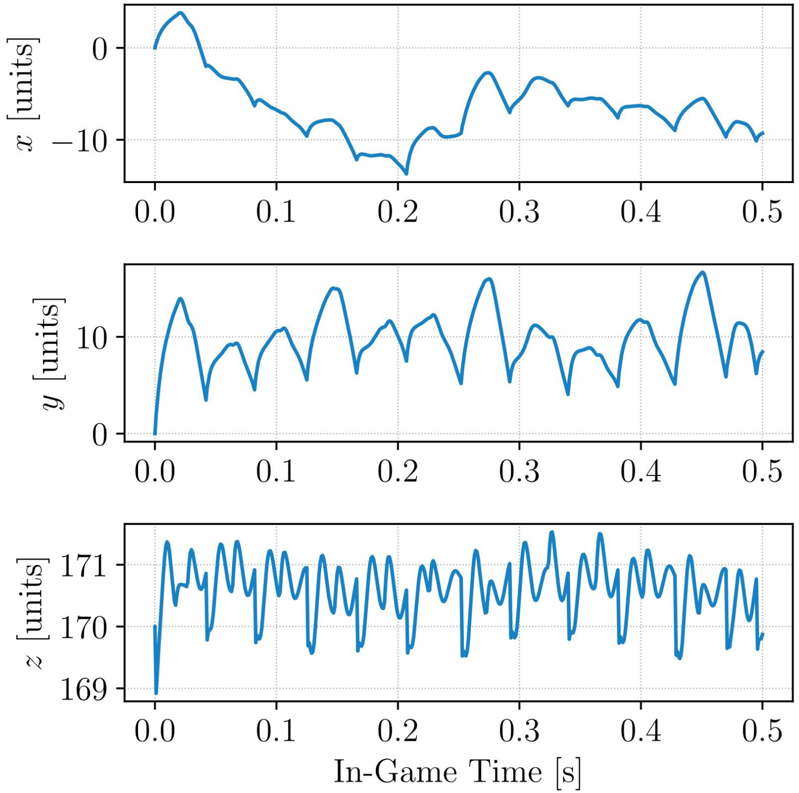
Results

Collision Accuracy Study

Collisions

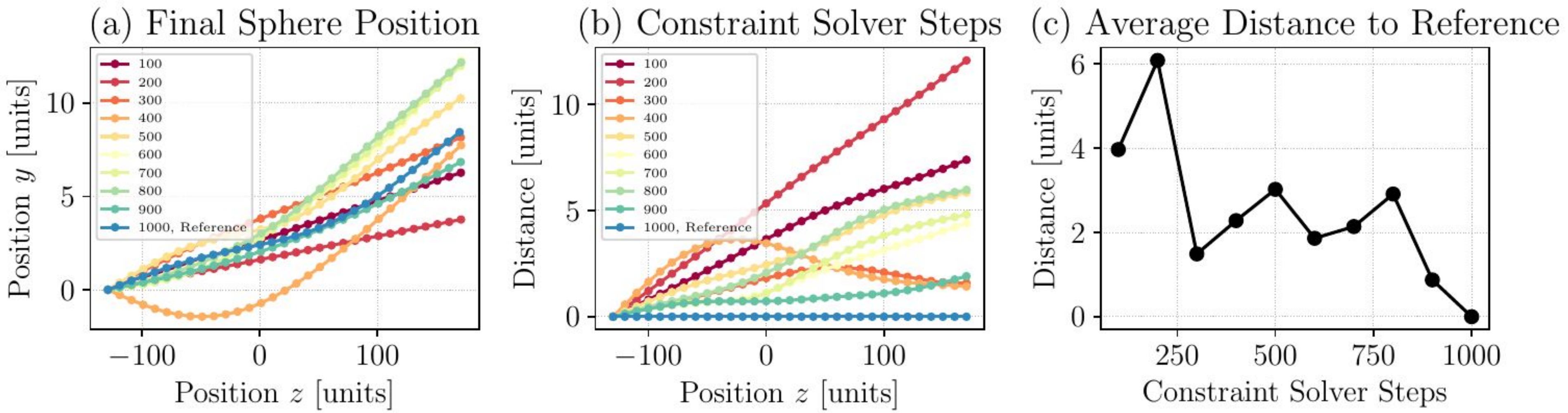
- Small longitudinal perturbation until collision with vessel wall





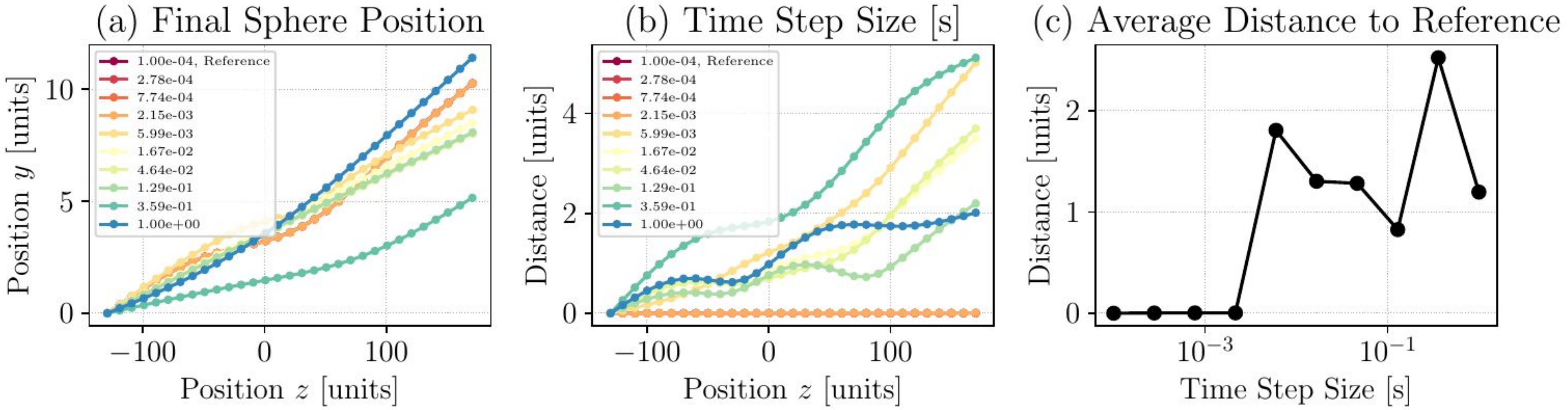
Collisions

Constraint Solver Steps



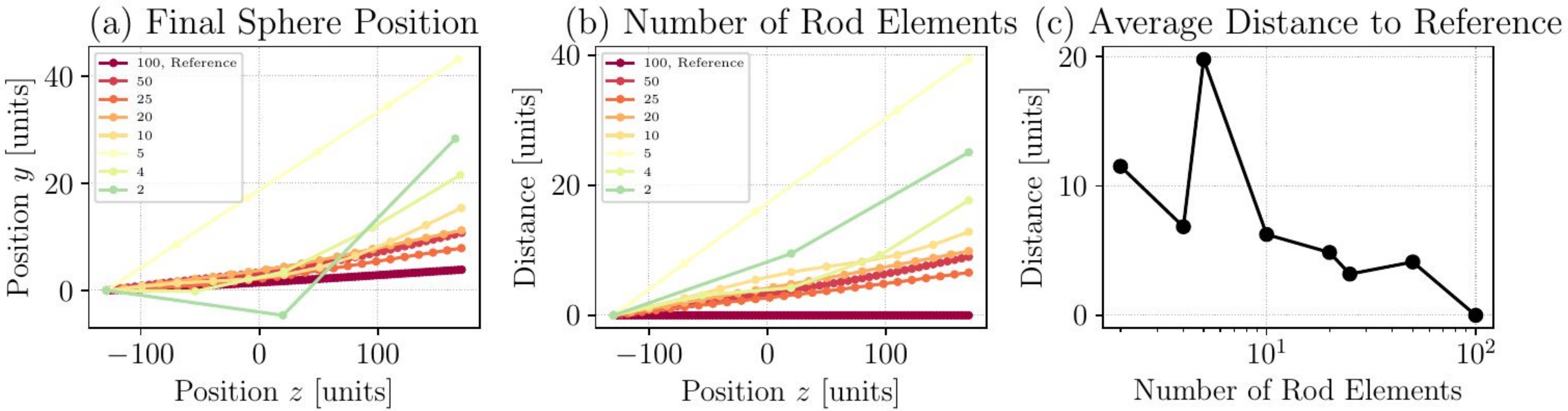
Collisions

Time Step Size



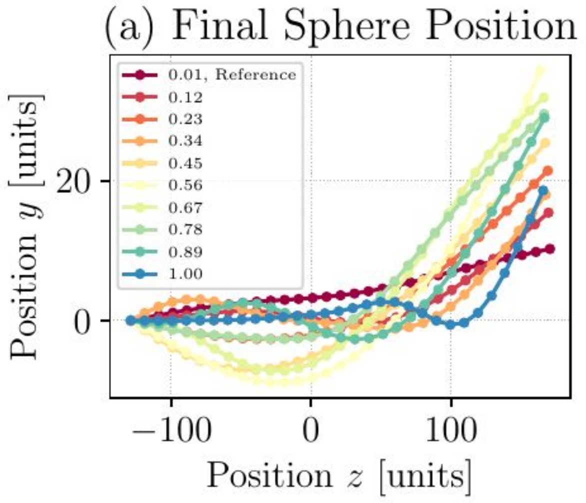
Collisions

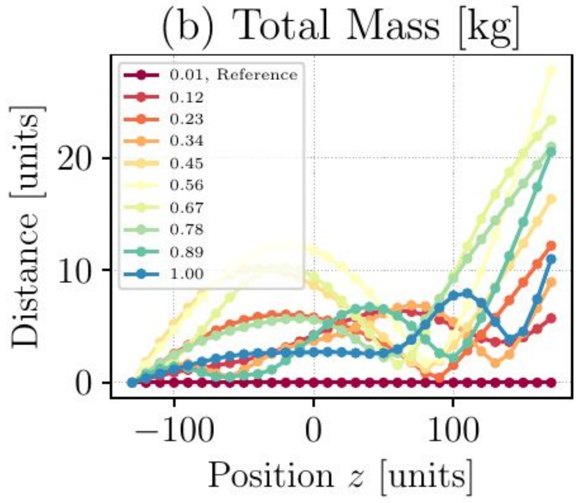
Number of Rod Elements

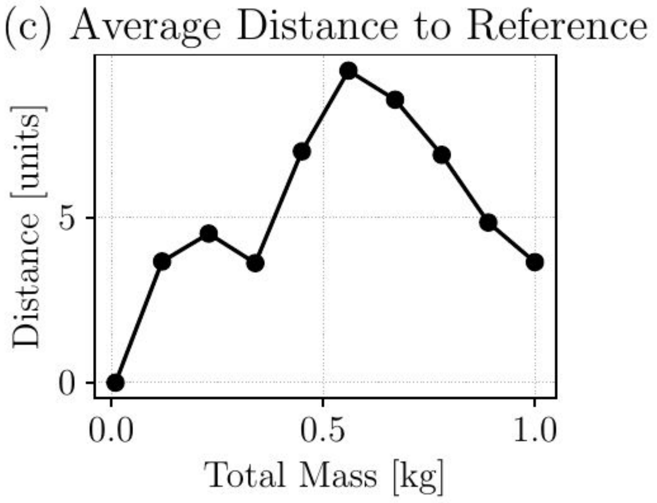


Collisions

Total Mass



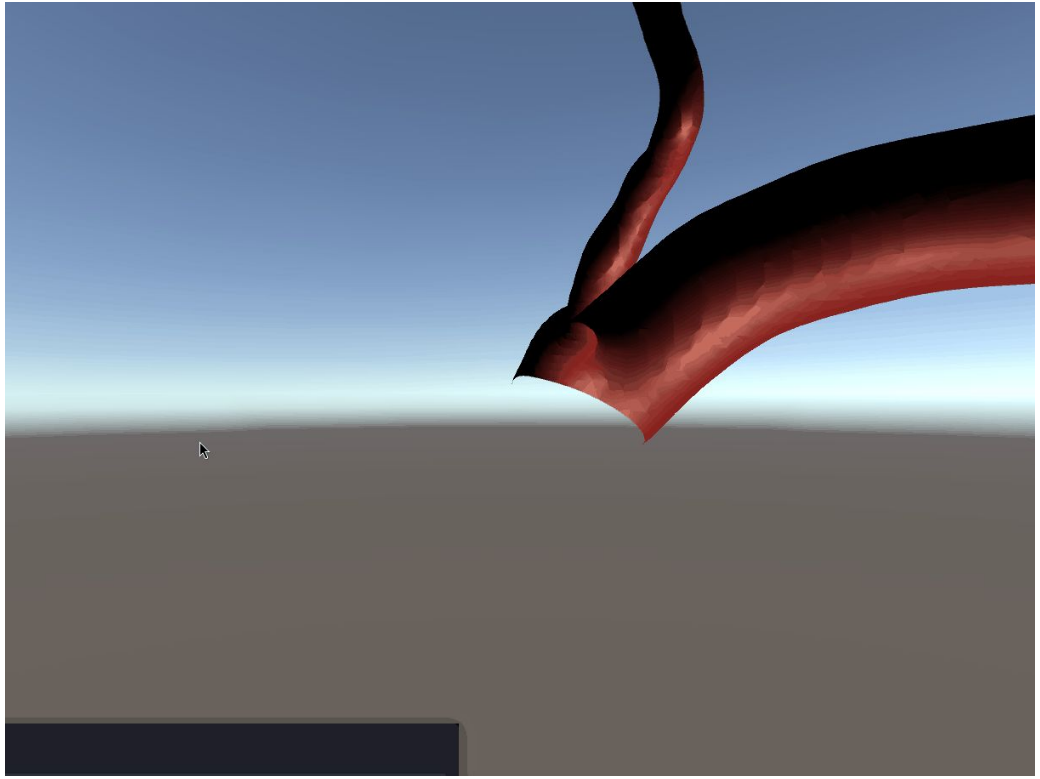




Results

Mouse Control

Mouse Control



Discussion

- Less rod elements do not affect accuracy much
  - But much less computation time
- Many parameters correlate with relaxation time / stiffness
  - Solution: Extended Position-Based Dynamics (XPBD)
- Verlet Integration makes it very stable and fast
  - Even high displacements do not break the simulation
- Runge-Kutta is not possible because derivatives can not be evaluated at future points

Conclusion

- Many new features for the improvement of the guidewire simulation in unity
- Enhanced software quality
- Framework for evaluating experiments with the guidewire
- Very stable and fast simulation with high physical accuracy possible