Eric Mouledoux

I.0: Requirements Documentation

I.1: Description of the Problem

**Name:** Cloth Physics

**Problem Statement:** Implement spring physic to simulate realistic cloth simulation.

**Problem Specification:** The application should simulate cloth using springs and physical forces such as gravity and wind.

I.2: Input Information

Description: Runtime interaction with the cloth

Type: N/A

Range of Acceptable Values: Mouse position and mouse buttons

I.3: Output Information

The program will simulate cloth based on input given before, and during runtime of the application.

I.4: User Interface Information

N/A

II.0: Design Documentation

II.1: System Architecture Description

The program is split into 3 different classes and 2 utilities. The main class of the simulation is Cloth.cs. This will create a grid of nodes, link the nodes with springs, and store apply post generation modifiers to specific variables. Node.cs stores unique values for each node in the system that will be used to move the node in space. Spring.cs links 2 game objects together and applies opposing forces to said game objects. The 2 utilities are CameraMove.cs and UI.cs. UI. cs is only used for exiting the program. CameraMove.cs is used to move the camera during runtime.

II.2: Information about the scripts

**File: Node.cs**

Class: Node

Data Members

Name: acl

Description: the current acceleration of this node in space

Name: vel

Description: the current velocity of this node in space

Name: frc

Description: the current force being applied to this node

Name: isLocked

Description: true or false if this node’s position is locked in spcae

Name: held

Description: true or false if the node is currently being manipulated by the user

Function: Awake

Parameters: none

Description: sets all initial values associated with this node to zero

Return: none

Function: FixedUpdate

Parameters: none

Description: checks for interaction by the user for dragging and locking nodes

Return: none

**File: Spring**

Class: Spring

Data Menbers

Name: springStrength

Description: the strength coefficent of the spring

Name: springLength

Description: the length of the spring at rest

Name: springDamp

Description: the damper coefficent of the spring

Name: node\_a

Description: a refrence to one of the object connect by the spring

Name: node\_b

Description: a refrence to one of the object connect by the spring

Name: a

Description: a refrence to the node component on node\_a

Name: b

Description: a refrence to the node component on node\_b

Name: Fs

Description: the force of the spring

Name: Fd

Description: the force of the spring damper

Name: e

Description: the direction the spring is facing to connent the 2 nodes

Name: Fg

Description: the force of gravity

Name: Ftotal

Description: the sum of all the forces

Name: strMod

Description: modifyable value to affect the spring's strength

Name: dmpMod

Description: modifyable value to affect the spring's damper strength

Name: grvMod

Description: modifyable value to affect the strength of gravity

Function: Build

Parameters: none

Description: creates a spring object between 2 pre-determined objects

Return: none

Function: FixedUpdate

Parameters: none

Description: runs all calculation functions and applies all forces to all connected nodes

Return: none

Function: CalculateSpringForce

Parameters: none

Description: calculates the spring force

Return: none

Function: CalculateSpringDamper

Parameters: none

Description: calculates the spring damper

Return: none

Function: CalculateNodeAceleration

Parameters: n(node)

Description: calculates the aceleration of a given node

Return: none

Function: CalculateNodeVelocity

Parameters: n(node)

Description: calculates the velocity of a given node

Return: none

**File: Cloth.cs**

Class: Cloth

Data Members

Name: node

Description: refrence to a prefabed node object

Name: spring

Description: refrence to a prefabed spring object

Name: strengthMod

Description: refrence to a UI slider for changing the spring strength at run time

Name: damperMod

Description: refrence to a UI slider for changing the spring damper strength at run time

Name: gravityMod

Description: refrence to a UI slider for changing the force of gravity at run time

Name: windForce

Description: the force of wind to be applied to the cloth

Name: rows

Description: the number of rows of nodes in the cloth

Name: columns

Description: the number of columns of nodes in the cloth

Name: offset

Description: the spacing between nodes in the cloth

Name: nodes

Description: a list of all the nodes in the cloth

Name: springs

Description: a list of all the springs in the cloth

Name: triangles

Description: a list of all the triangles in the cloth

Function: Start

Parameters: none

Description: calls the MakeCloth function at the start of the program

Return: none

Function: MakeCloth

Parameters: none

Description: creates a grid of nodes and connects them with springs to create triangles

Return: node

**File: CameraMove.cs**

Class: CameraMove

Data Members

Name: speed

Description: the speed the camera moves, zooms, and rotates at

Name: resetKey

Description: the key the user can press to retes the camera to its original position

Name: originPos

Description: the original position of the camera

Name: originRoe

Description: the original rotation of the camera

Name: originScl

Description: the original scale of the camera

Function: Awake

Parameter: none

Description: sets the original position of the camera

Return: none

Function: Update

Parameter: none

Description: keeps track of mouse buttons and movement for moving the camera in space

**File UI.cs**

Class: UI

Data Members

Function: Exit

Parametr: none

Description: closes the application

Return: none

Function: Restart

Parameter: none

Description: reloads the current scene

Return: none

III.0: Implementation Documentation

III.1 Program Code

**File: Node.cs**

using UnityEngine;

using System.Collections;

public class Node : MonoBehaviour

{

public Vector3 acl;

public Vector3 vel;

public Vector3 frc;

public bool isLocked;

bool held;

void Awake()

{

acl = vel = frc = Vector3.zero;

}

void FixedUpdate()

{

Vector3 mouseSpace = Input.mousePosition - new Vector3(Screen.width / 2,

Screen.height / 2,

1).normalized;

Vector3 screenMid = (Camera.main.transform.position +

Camera.main.transform.forward \*

Vector3.Distance(transform.position,

Camera.main.transform.position));

if(Input.GetMouseButtonDown(0))

{

if(Vector3.Distance(screenMid, transform.position) < 10)

{

held = true;

}

}

else if(Input.GetMouseButtonUp(0))

{

held = false;

}

if(held)

{

acl += (screenMid - transform.position);

if(Input.GetKeyDown(KeyCode.L))

{

isLocked = !isLocked;

if(isLocked)

{

acl = vel = frc = Vector3.zero;

}

}

}

}

}

**File: Spring.cs**

using UnityEngine;

using System.Collections;

public class Spring : MonoBehaviour

{

public void Build ()

{

Ftotal = Vector3.zero;

Fg = new Vector3(0, -1f, 0);

strMod = dmpMod = grvMod = 1.0f;

if(node\_a && node\_b)

{

if(!node\_a.GetComponent<Node>())

node\_a.AddComponent<Node>();

if(!node\_b.GetComponent<Node>())

node\_b.AddComponent<Node>();

a = node\_a.GetComponent<Node>();

b = node\_b.GetComponent<Node>();

springLength = Vector3.Distance(a.transform.position, b.transform.position);

}

}

void FixedUpdate ()

{

Fg = new Vector3(0, -1f, 0) \* grvMod;

/////////////////////////////////////////////////////////////

e = (b.transform.position - a.transform.position).normalized;

CalculateSpringForce();

CalculateSpringDamper();

Ftotal = (Fs \* e) + (Fd \* e);

CalculateNodeAcceleration(a);

Ftotal = -Ftotal; // Q-Dogs assistants

CalculateNodeAcceleration(b);

CalculateNodeVelocity(a);

CalculateNodeVelocity(b);

node\_a.transform.position += a.vel;

node\_b.transform.position += b.vel;

GetComponent<LineRenderer>().SetPosition(0, node\_a.transform.position);

GetComponent<LineRenderer>().SetPosition(1, node\_b.transform.position);

transform.position = (a.transform.position + b.transform.position) / 2;

if(Vector3.Distance(a.transform.position, b.transform.position) > springLength \* 3)

{

Destroy(gameObject);

}

a.acl = Vector3.zero;

b.acl = Vector3.zero;

}

void CalculateSpringForce()

{

Fs = -(springStrength \* strMod) \* (springLength - Vector3.Magnitude(b.transform.position - a.transform.position));

}

void CalculateSpringDamper()

{

float aVel = Vector3.Dot(e, a.vel);

float bVel = Vector3.Dot(e, b.vel);

Fd = -(springDamp \* dmpMod) \* (aVel - bVel);

}

void CalculateNodeAcceleration(Node n)

{

n.acl += (Ftotal + Fg) \* Time.fixedDeltaTime;

}

void CalculateNodeVelocity(Node n)

{

//if it's locked zero else add acceleration \* dt

n.vel += !n.isLocked ? n.acl \* Time.fixedDeltaTime : Vector3.zero;

}

public float springStrength;

public float springLength;

public float springDamp;

public GameObject node\_a;

public GameObject node\_b;

Node a, b;

float Fs;

float Fd;

Vector3 e;

Vector3 Fg;

Vector3 Ftotal;

public float strMod;

public float dmpMod;

public float grvMod;

}

**File: Cloth.cs**

using UnityEngine;

using UnityEngine.UI;

using System.Collections.Generic;

public class Cloth : MonoBehaviour

{

void Start()

{

MakeCloth();

}

void FixedUpdate()

{

foreach(GameObject s in springs)

{

if (s)

{

s.GetComponent<Spring>().strMod = strengthMod.value;

s.GetComponent<Spring>().dmpMod = damperMod.value;

s.GetComponent<Spring>().grvMod = gravityMod.value;

}

else

{

springs.Remove(s);

}

}

if(Input.GetKey(KeyCode.Space))

{

windForce += Camera.main.transform.forward \* Time.fixedDeltaTime;

foreach(List<GameObject> t in triangles)

{

Vector3 wind = Vector3.zero;

Vector3 velocity = Vector3.zero;

Vector3 norm = Vector3.zero;

float area = 0;

velocity = t[0].GetComponent<Node>().vel +

t[1].GetComponent<Node>().vel +

t[2].GetComponent<Node>().vel;

velocity /= 3;

norm = Vector3.Cross(t[0].transform.position.normalized,

t[1].transform.position.normalized);

area = 0.5f \*

(Vector3.Distance(t[0].transform.position, t[1].transform.position) \*

Vector3.Distance(t[0].transform.position, t[2].transform.position));

wind = velocity.magnitude \* windForce;

foreach(GameObject n in t)

{

n.GetComponent<Node>().acl += wind / 3;

}

}

}

else if (windForce.magnitude > 0)

{

windForce -= windForce \* Time.fixedDeltaTime;

}

}

/// <summary>

/// Makes the cloth.

/// </summary>

void MakeCloth()

{

// stops time to prevent calculations before the gris is made

Time.timeScale = 0;

// the current node we are making

Vector2 key;

// the position for the node to be

Vector3 nextPos = transform.localPosition;

// for every column

for(int i = 0; i < columns; i++)

{

// and for every row

for(int j = 0; j < rows; j++)

{

// make a node at the next position

GameObject n = Instantiate(node, nextPos, transform.localRotation) as GameObject;

// and rename it to its grid position [x, y] - [col, row]

n.name = "[" + i + "][" + j + "]";

// assign the current key to the current node

key = new Vector2(i, j);

// add the key to the dictionary of nodes (vec2, gameobject)

nodes.Add(key, n);

Vector2 s1 = new Vector2(i - 1, j); // the node to the left

Vector2 s2 = new Vector2(i, j - 1); // the node below

Vector2 s3 = new Vector2(i - 1, j - 1); // the node below and to the left

Vector2 s4 = new Vector2(i - 1, j + 1); // the node above and to the left

/\* c = current node [1, 1]

\* s1 = [0, 1]

\* s2 = [1, 0]

\* s3 = [0, 0]

\* s4 = [0, 2]

\* \*\*\*\*\*\*\*\*\*\*\*\*\*

\* s4 \ 0 0

\* \

\* s1 -- c 0

\* / |

\* s3 / s2 0

\*/

// if any of the nodes exist connect it to this one with a spring

if(nodes.ContainsKey(s1))

{

GameObject s = Instantiate(spring);

s.GetComponent<Spring>().node\_a = nodes[key];

s.GetComponent<Spring>().node\_b = nodes[s1];

s.GetComponent<Spring>().Build();

s.transform.parent = nodes[key].transform;

springs.Add(s);

}

if(nodes.ContainsKey(s2))

{

GameObject s = Instantiate(spring);

s.GetComponent<Spring>().node\_a = nodes[key];

s.GetComponent<Spring>().node\_b = nodes[s2];

s.GetComponent<Spring>().Build();

s.transform.parent = nodes[key].transform;

springs.Add(s);

}

if(nodes.ContainsKey(s3))

{

GameObject s = Instantiate(spring);

s.GetComponent<Spring>().node\_a = nodes[key];

s.GetComponent<Spring>().node\_b = nodes[s3];

s.GetComponent<Spring>().Build();

s.transform.parent = nodes[key].transform;

springs.Add(s);

}

if(nodes.ContainsKey(s4))

{

GameObject s = Instantiate(spring);

s.GetComponent<Spring>().node\_a = nodes[key];

s.GetComponent<Spring>().node\_b = nodes[s4];

s.GetComponent<Spring>().Build();

s.transform.parent = nodes[key].transform;

springs.Add(s);

}

// locks the outer edges

if(/\*j == 0 ||\*/ j == rows -1 )//|| i == 0 || i == columns - 1)

nodes[key].GetComponent<Node>().isLocked = true;

// increase the next position by an offset value

nextPos += transform.up \* offset;

}

// increase the next position by another offset (side-by-side)

nextPos = transform.position;

nextPos.x += offset \* (i + 1);

// reset the offet to the bottom most point

nextPos.y = transform.position.y;

}

//GameObject bottom = Instantiate(spring);

//bottom.GetComponent<Spring>().node\_a = nodes[Vector2.zero];

//bottom.GetComponent<Spring>().node\_b = nodes[new Vector2(columns-1, 0)];

//bottom.GetComponent<Spring>().Build();

// Makes the tirangles

// for every column

for(int i = 0; i < columns; i++)

{

// and for every row

for(int j = 0; j < rows; j++)

{

if(i % 2 == 0)

{

List<GameObject> n = new List<GameObject>();

Vector2 up = new Vector2(i, j+1);

Vector2 right = new Vector2(i+1, j);

if(nodes.ContainsKey(up) &&

nodes.ContainsKey(right))

{

n.Add(nodes[new Vector2(i, j)]);

n.Add(nodes[up]);

n.Add(nodes[right]);

triangles.Add(n);

}

}

else

{

List<GameObject> n = new List<GameObject>();

Vector2 down = new Vector2(i, j-1);

Vector2 left = new Vector2(i-1, j);

if(nodes.ContainsKey(down) &&

nodes.ContainsKey(left))

{

n.Add(nodes[new Vector2(i, j)]);

n.Add(nodes[down]);

n.Add(nodes[left]);

triangles.Add(n);

}

}

}

}

// restart time

Time.timeScale = 1;

}

public GameObject node;

public GameObject spring;

public Slider strengthMod;

public Slider damperMod;

public Slider gravityMod;

Vector3 windForce;

public int rows;

public int columns;

public float offset;

Dictionary<Vector2, GameObject> nodes = new Dictionary<Vector2, GameObject>();

List<GameObject> springs = new List<GameObject>();

List<List<GameObject>> triangles = new List<List<GameObject>>();

}

**File: CameraMove.cs**

using UnityEngine;

using System.Collections;

[RequireComponent (typeof (Camera))]

/// <summary>

/// Addes the ability to move and rotate the camera that this script is attached to.

/// </summary>

public class CameraMove : MonoBehaviour

{

void Awake()

{

// Saves the initial transform values for later resetting

originPos = transform.position;

originRot = transform.localEulerAngles;

originScl = transform.localScale;

}

void Update ()

{

float Th = 0; // Horizontal transform value

float Tv = 0; // Verticle transform value

float Rx = 0; // Rotation around the X-axis

float Ry = 0; // Rotation around the Y-axis

if(Input.GetMouseButton(2)) // Checks if the left mouse button is being clicked

{

Th = speed \* Input.GetAxis("Mouse X") \* Time.deltaTime; // Sets Th to the difference the horizontal movement of the mouse

Tv = speed \* Input.GetAxis("Mouse Y") \* Time.deltaTime; // Sets Tv to the difference the verticle movement of the mouse

}

if(Input.GetMouseButton(1)) // Checks if the right mouse button is being clicked

{

Rx = speed \* 0.25f \* Input.GetAxis("Mouse Y") \* Time.deltaTime; // Sets Rx to the difference the verticle movement of the mouse

Ry = speed \* 0.25f \* Input.GetAxis("Mouse X") \* Time.deltaTime; // Sets Ry to the difference the horizontal movement of the mouse

}

if(Input.GetAxis("Mouse ScrollWheel") > 0) // Checks if the mouse wheele is being rolled forward

transform.position += transform.forward \* speed \* Time.deltaTime; // Moves the camera forward

if(Input.GetAxis("Mouse ScrollWheel") < 0) // Checks if the mouse wheele is being rolled backward

transform.position -= transform.forward \* speed \* Time.deltaTime; // Moves the camera backward

transform.Translate(-Th, -Tv, 0); // Moves the camera: left, right, up, and down

transform.RotateAround(transform.position, transform.right, -Rx); // Rotates the camera to look up

transform.RotateAround(transform.position, Vector3.up, Ry); // Rotates the camera to look down

/// Will reset the camera ti its initial position //

if(Input.GetKey(resetKey)) //

{ //

transform.position = originPos; //

transform.localEulerAngles = originRot; //

transform.localScale = originScl; //

}

}

public float speed; // The speed the camera will move or rotate at

public KeyCode resetKey; // The key that will be used to reset the camera's transform

Vector3 originPos; // Initial position of the camera

Vector3 originRot; // Initial rotation of the camera

Vector3 originScl; // Initial scale of the camera

}

**File: UI.cs**

using UnityEngine;

using System.Collections;

public class UI : MonoBehaviour

{

public void Exit()

{

Application.Quit();

}

public void Restart()

{

Application.LoadLevel(Application.loadedLevel);

}

}

Source Code: <https://github.com/Mouledoux/Physics>

IV.0: Verification and Validation Documentation

IV.1: Test Plan

IV.2: Operating Directions

To run this program navigate to the "Physics" folder and run Physics.exe

Hold MMB to pan the camera

Hold RMB to rotate the camera

Hold LMB to drag sections of the cloth

Tap "L" to lock or unlock cloth segments

Hold space to make wind