**Name: Kuan-Ting Chin**

**Student ID: 33430072**

**Physics Assignment: Physics Project with FFL**

**Aim: To create a simulation of cloth in real time in Unity.**

**Video link - https://youtu.be/mHykYWW23Sw**

**Objective:**  
The main objective of the physics project was to create cloth simulation in either Unity or Octet. This project could be done in a group of 2, but I decided to do it by myself in the end.

The main focus of cloth simulation is to understand that the simulation is actually only simulating the mass of particles and its relevant connections, usually through constraints or springs. This in return allows us to perceive them as fibres in the material for cloth in the real world.

**Implementation:**

The implementation of the cloth was to first create cloths nodes, simply done via particles. It would be creating lists of constraints, and public classes which gave individual particles variables that could later on be linked together to be affected by external forces potentially.

One of the most important functions in the constraint script, is the constraintsMet function.  
  
public void constraintsMet(bool debug = false)

{

if (p1.deadParticles || p2.deadParticles)

return;

if (isTorn)

{

if (debug)

Debug.DrawLine(p1.getPos(), p2.getPos(), Color.black);

return;

}

Vector3 p1\_to\_p2 = p2.getPos() - p1.getPos();

if (Mathf.Abs(p1\_to\_p2.magnitude - particleDiff) < 0.005f) return;

float current\_distance = p1\_to\_p2.magnitude;

Vector3 correctionVector = p1\_to\_p2 \* (1 - particleDiff / current\_distance);

Vector3 correctVectorHalf = correctionVector \* 0.5f;

p1.offsetPos(correctVectorHalf);

p2.offsetPos(-correctVectorHalf);

}

This function allows the particles p1 and p2 (declared as in the clothparticles script) constraint to be solved. The method solved is by the ClothParticles.timeStep() function. The timeStep function:

public void timeStep()

{

if (move && !deadParticles)

{

Vector3 temp = currentPos;

currentPos = currentPos + (currentPos - previousPos) \* (1.0f - parent.damping) + acceleration;

previousPos = temp;

acceleration = Vector3.zero;

int aliveConstraints = 0;

foreach (Constraint c in constraints)

if (c.isTorn == false)

aliveConstraints++;

if (aliveConstraints == 0)

deadParticles = true;

}

}

Thoroughly checks all constraints through iterating each and every constraint, then calculate its relevant position in the particle.

The scriptCloth script, the main particle handling script, contains lists of particles, constraints, widths and height of the cloth and others.

The main method of creating the cloth, is to create a vector which stores the particle nodes in a grid like system (Thanks to JP Evette).

particles = new clothParticles[nodeWidth \* nodeHeight]; //Vector to store matrices of node positions.

for (int x = 0; x < nodeWidth; x++)

{

for (int y = 0; y < nodeHeight; y++)

{

Vector3 pos = new Vector3(width \* (x / (float)nodeWidth), -height \* (y / (float)nodeHeight), 0);

particles[y \* nodeWidth + x] = new clothParticles(this, pos); // insert particle xth column of Y row

}

}

Once the nodes are stored, it is important to set the constraints (with the public virtual functions set in script previously) in order to link the nodes together.

// Connecting close proximity neighbours with constraints.

for (int x = 0; x < nodeWidth; x++)

{

for (int y = 0; y < nodeHeight; y++)

{

if (x < nodeWidth - 1) makeConstraint(getParticle(x, y), getParticle(x + 1, y));

if (y < nodeHeight - 1) makeConstraint(getParticle(x, y), getParticle(x, y + 1));

if (x < nodeWidth - 1 && y < nodeHeight - 1) makeConstraint(getParticle(x, y), getParticle(x + 1, y + 1));

if (x < nodeWidth - 1 && y < nodeHeight - 1) makeConstraint(getParticle(x + 1, y), getParticle(x, y + 1));

}

}

// Connecting secondary neighbours with constraints.

for (int x = 0; x < 3; x++)

{

for (int y = 0; y < nodeHeight; y++)

{

if (x < nodeWidth - 2) makeConstraint(getParticle(x, y), getParticle(x + 2, y));

if (y < nodeHeight - 2) makeConstraint(getParticle(x, y), getParticle(x, y + 2));

if (x < nodeWidth - 2 && y < nodeHeight - 2) makeConstraint(getParticle(x, y), getParticle(x + 2, y + 2));

if (x < nodeWidth - 2 && y < nodeHeight - 2) makeConstraint(getParticle(x + 2, y), getParticle(x, y + 2));

}

}

Once done, it is important we ensure the flag is attached to the ‘pole’ to form the flag, and allow our simulation to work.

//attaching node to the pole, relative to the size of the flags size.

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

{

getParticle(0 + i, 0).offsetPos(new Vector3(0.8f, 0.7f, 0.7f));

getParticle(0 + i, 0).makeUnmovable();

getParticle(nodeWidth - 1 - i, 0).offsetPos(new Vector3(-0.8f, 0.7f, 0.7f));

if (nodeUnmovable)

{

getParticle(0, nodeHeight - 1 - i).offsetPos(new Vector3(0.8f, 0.7f, 0.7f));

getParticle(0, nodeHeight - 1 - i).makeUnmovable();

getParticle(nodeWidth - 1 - i, nodeHeight - 1 - i).offsetPos(new Vector3(-0.8f, 0.7f, 0.7f));

getParticle(nodeWidth - 1 - i, nodeHeight - 1 - i).makeUnmovable();

}

}

The code above just ensures the flags are connected at the top of cloth, whilst the bottom nodes are free to move within constraint limitations.

Even if the particles were created, it would require meshes to be rendered for the user to see, thus a meshCreator script is needed. The meshCreator script contains 2 meshfilters, the cloth mesh strings (so as to see the cloth folds better) as well as the cloth mesh itself:

The meshCreator script also has the function calcTriangleNormals:

Vector3 calcTriangleNormal(clothParticles p1, clothParticles p2, clothParticles p3)

{

Vector3 pos1 = p1.getPos();

Vector3 pos2 = p2.getPos();

Vector3 pos3 = p3.getPos();

Vector3 v1 = pos2 - pos1;

Vector3 v2 = pos3 - pos1;

return Vector3.Cross(v1, v2);

}

Which simply calculates the normal of the triangle, with the use of Unity’s cross product in order to fill in the fragment shader. The drawStrings function:

void drawStrings()

{

int index = 0;

for (int i = 0; i < cloth.constraints.Count; i++)

{

if (cloth.constraints[i].IsConstraintDead())

continue;

vertices.Add(cloth.constraints[i].GetPositions()[0]);

vertices.Add(cloth.constraints[i].GetPositions()[1]);

indices.Add(index++);

indices.Add(index++);

}

viewMeshStrings.Clear();

viewMeshStrings.vertices = vertices.ToArray();

viewMeshStrings.SetIndices(indices.ToArray(), MeshTopology.Lines, 0);

viewMeshStrings.RecalculateBounds();

}

Does the same thing, except the mesh is focused on the vertex shader instead.

A small thing added, was to create a slider GUI for the user to adjust the simulation of wind forces along its x,y and z axis, and is reflected upon via the movementForce inside the scriptCloth script.

**Conclusion**

I felt this project was challenging, and sometimes a little hard to understand, but it was enjoyable to see something so complex come to life. If I had more knowledge and time, I would have focused a bit more on integrating collision with the particles, in order to simulate the cloth wrapping around an object or such.