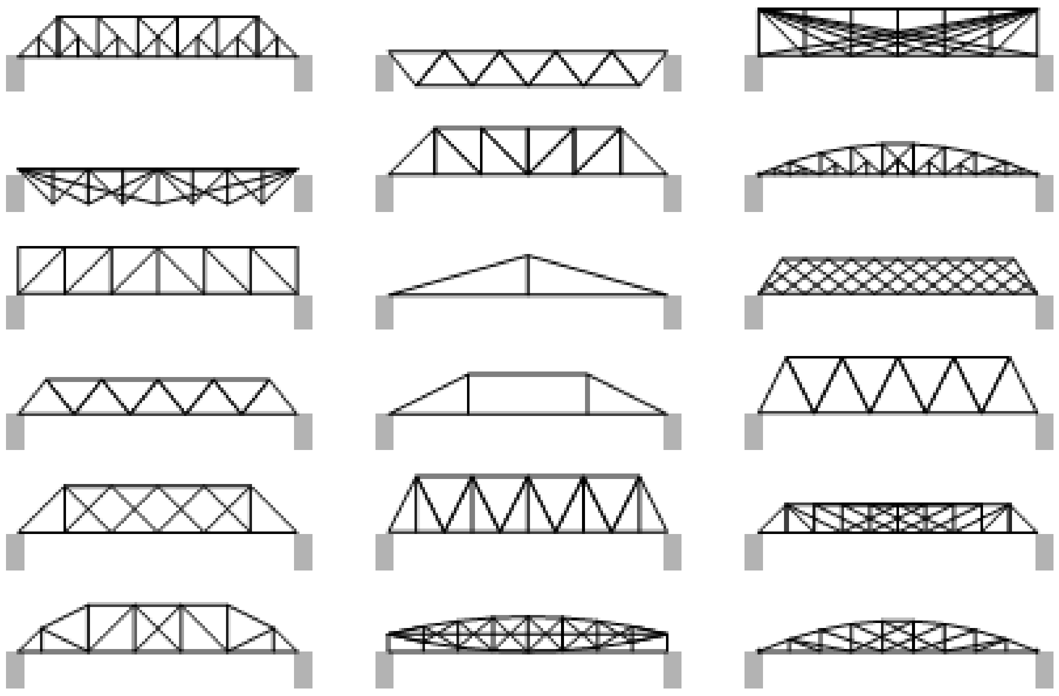
Project Truss Bridges

Introduction

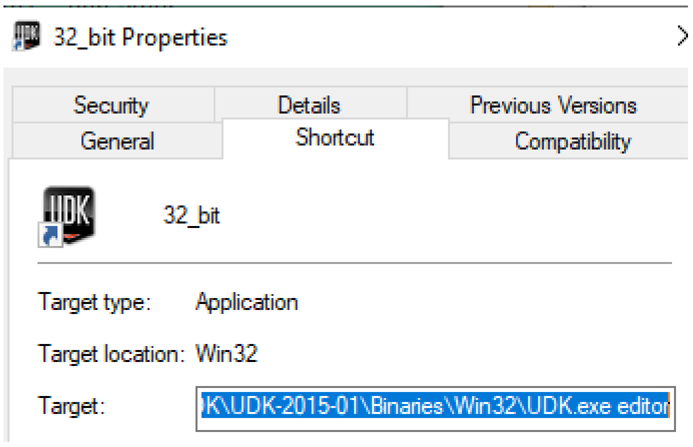
There are various ways to design bridges. Most modern bridges are designed using advanced modelling and simulation software which we do not have access to. Yet the design principles used today are based upon the most efficient use of materials, while achieving an aesthetic appealing and sexy outcome. In the Victorian era, attention to sexy aesthetics was not the issue, the most efficient use of materials which could produce a viable structure was paramount. Truss bridges were developed in this era, to make the best use of iron. These bridges were designed as a series of straight iron bar *links* joined together in connected joints or *nodes*. The diagrams below show some approaches to spanning a gap. Each of these structures has a name which you can research.



Running the Engine

The truss is contained in the level **SciencePark_Level_Truss.udk**. Because it makes use of some c-code you must run UDK in 32-bit mode.

1) Navigate here **C:\UDK\UDK-2015-01\Binaries\Win32**
2) Right-click on **UDK.exe**
3) Choose Send to > Desktop.
4) Right click on icon and select Properties. Add editor to the end of the Target:


32_bit Properties

Security | Details | Previous Versions
General | Shortcut | Compatibility

32_bit

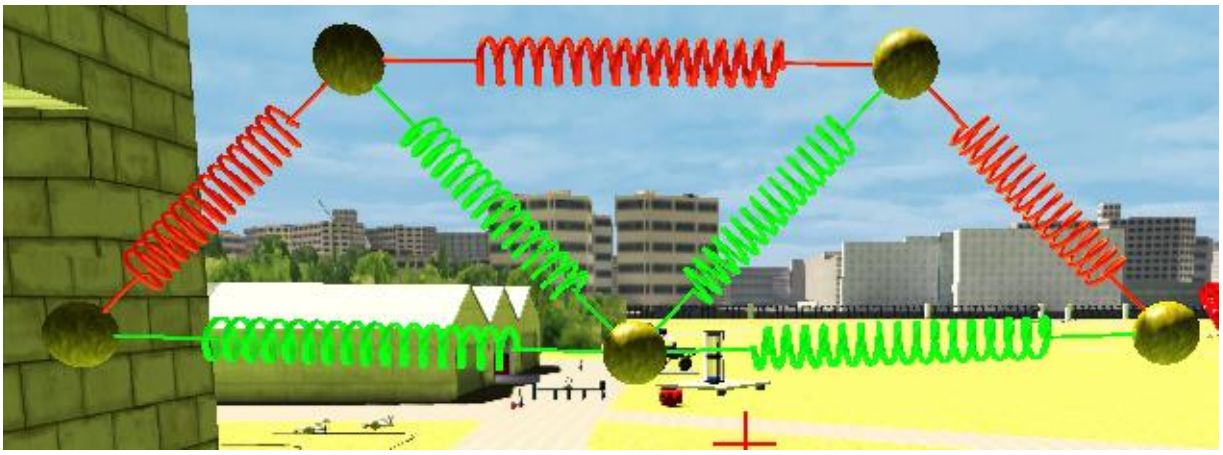
Target type: Application

Target location: Win32

Target: K\UDK-2015-01\Binaries\Win32\UDK.exe editor

MAS14 Simulation Engine

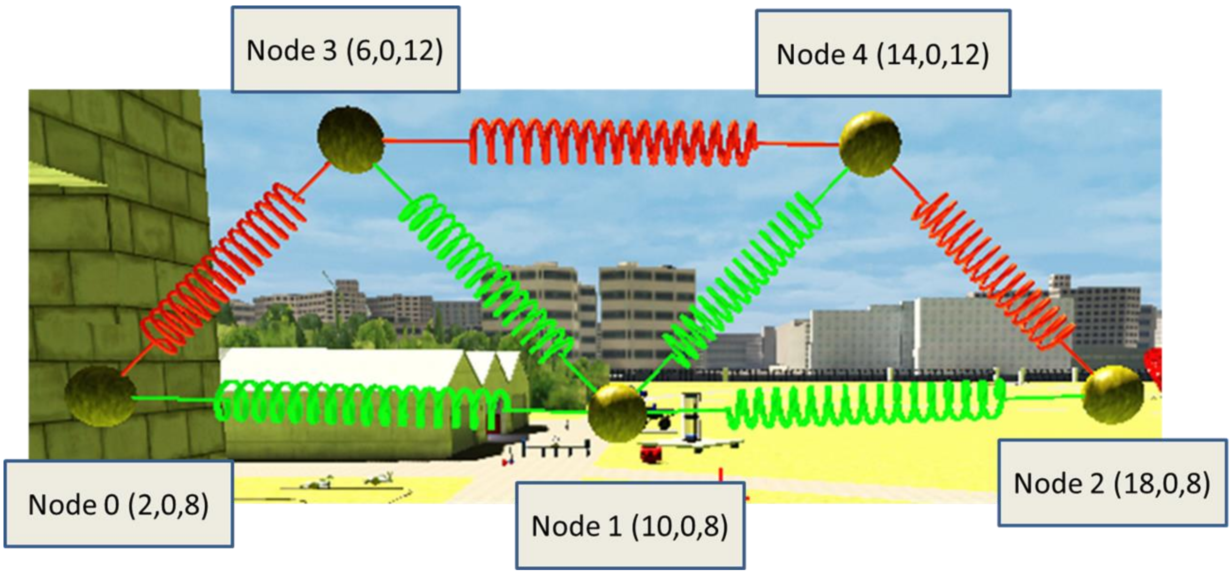
This simulation engine has been developed to allow you to investigate various Truss Bridge designs and to compare various designs. Unlike the solid iron trusses shown above, the simulation engine represents each iron link as a spring, and each connection is represented as a ball. The engine therefore magnifies the deflection of each member so you can see whether the member is in compression (red) or extension (green) or else does not bear any load (grey). The figure below shows a 'Warren' truss where the left and right bottom nodes are constrained not to move (they could be on pillars). The centre bottom node has a load force pulling downwards. You can see which link is under compression or extension, and this should make sense.



Coding the Truss

You can design any truss of your choice. Previous students have chosen 2D trusses, but the engine should be capable of 3D trusses.

Trusses are coded using our special scripting language. An example of a .txt file to do this is shown below. Each line of the file starts with a letter which tells the interpreter what the following numbers mean. The truss shown above was created using this file. The diagram below shows the labelling of the nodes.



Here each node is labelled with the (X,Y,Z) values. These are in metres.

```
N   2 0   8 0
N 10 0   8 0
N 18 0   8 0
N   6 0  12 0
N 14 0  12 0

L 0 1 0 0
L 0 3 0 0
L 1 0 0 0
L 1 2 0 0
L 1 3 0 0
L 1 4 0 0
L 2 1 0 0
L 2 4 0 0
L 3 0 0 0
L 3 1 0 0
L 3 4 0 0
L 4 1 0 0
L 4 2 0 0
L 4 3 0 0

C 0 1 1 1
C 2 1 1 1



F 1 0 0 -10
E 4 4 4 4
```

Lines starting with **N**
These locate the **nodes** in x-z space e.g the first line locates a node at x = 2, y = 0 z = 8. The final 0 is not used.
Each line declares a node, starting at index 0. So here we have nodes 0, 1, 2, 3, 4

Lines starting with **L**
These establish **links** between nodes. So the first line indicates that node 0 is connected to node 1. The second line shows that node 0 is also connected to node 3. The third line shows that node 1 is connected to node 0. **It is important that if node A is connected to node B, then you must connect node B to node A.**

Lines starting with **C**
These are the **constraints** on the nodes, in other words those nodes which cannot move. These are usually the pillars supporting the ends of the bridge. The line “C 0 1 1 1” means that node 0 cannot move in the X,Y,Z directions. It is fixed to a pillar and to the ground. Same for node 2

Lines starting with **F**
These are the loads or **forces** applied to nodes. In this case the line “F 1 0 0 -10” means that node 1 experience a force Fx=0, Fy=0, Fz=-10. This is not really necessary since forces can be applied at run-time.

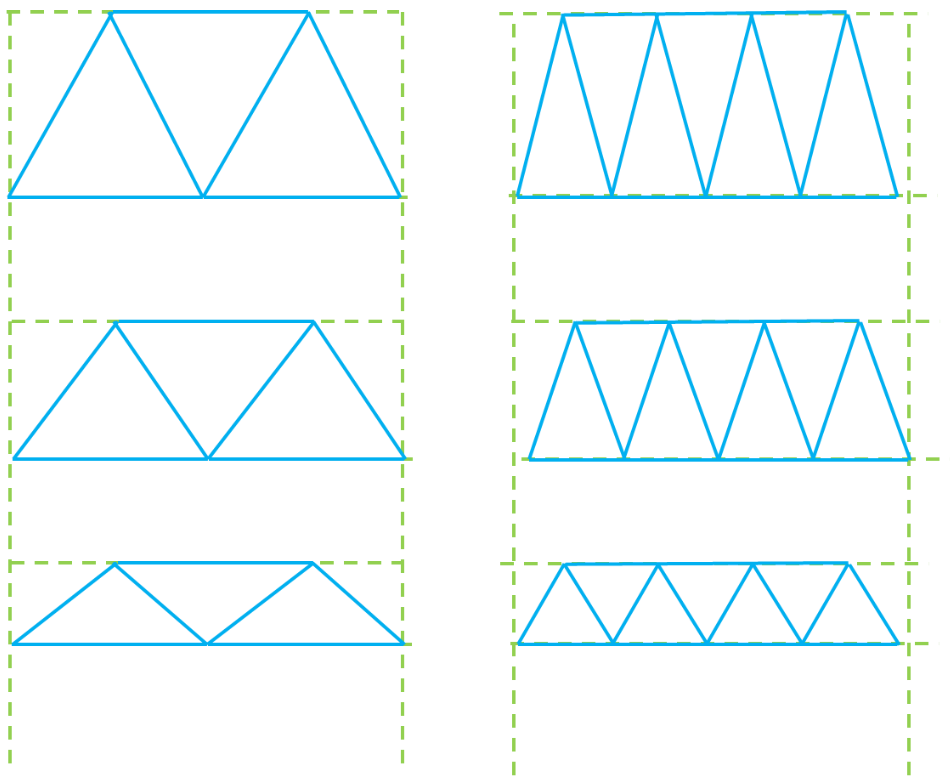
Line starting with **E.** Indicates the **end** of the design file

What and How to Investigate

1) The most obvious thing to do is to design your own truss bridge and subject this to investigation. Better to compare two truss bridges which have something in common, e.g., they both span the same space between the banks of a river. Here's a way of comparing trusses that span the same gap.

Going down the columns, the only thing that changes is height of the structure. Going from left to right, the extra bars are added into each structure.

So a typical investigation would be to apply a series of loads to the centre bottom node and note down the deflexion for each load. Then plot this as a graph. Repeat this for other truss designs and make a comparison.



2) By clicking on any node and using **setParam load 10** you can change the load to a range of values, and you can measure the X,Y,Z disp (displacement) by reading the HUD. An engineer would be interested in plotting a graph of load (independent variable – along the bottom) and displacement (dependent variable – up the side).
3) Also you could look at the coloured springs. You will see some patterns here depending on your truss design. Remember green is in tension and red is in compression.

4) Remember, when you click on a link, it will show you the forces in the link. So you could look at these forces, and see which ones are large and which ones are small. This will show you which links are carrying most of the load.

5) You may notice that when you change a load, then the structure will oscillate (vibrate). If you want to investigate vibrations, then you can slow them down by increasing the node mass.

Changing the Code

1) I need to see some code in your report (look at the grading matrix). A straightforward way to do this would be to look at the User Interface, including the HUD. If you select a node, apply a load and look at the HUD you will see a lot of things displayed which are not being used. This is because I was researching this project. A good idea would be to edit **MAS14_Node.uc** and delete lines which are not being used. You might also like to change the colors used. Then you could include your cleaned-up HUD code in your report to attract credit.

2) You could also extend the **SetParam** function in **MAS14_Node.uc** . Currently this looks like this

```
function SetParam(string paramName,float paramValue){
  WorldInfo.Game.Broadcast(self,paramName@paramValue);

  paramName = Caps(paramName); //capitalize so case insensitive

  if(paramName == Caps("load")){
           self.Load.Z = paramValue;
  }

}
```

You see that the console command **setParam load 10** will set **self.load.Z** to 10. You could add another if block to set the X or Y load to a value.

3) I have some other suggestions. Read on

Cleaning up the Truss code

Problem is that the force calculation **MAS14P_TrussConfig.uc** line 350 does not refer to the link. So when the link is selected at run-time and its k1 or k3 are changed, nothing happens, since these values are not being pulled down from the link. This is bad bad OO design.
So to fix, we need to grab the link connecting the two nodes **thisNode** and **linkNode** in the inner loop line 336-353. In other words we need something like this, in place of line 350.

```
          ext = (len - lenOrig);
          findLink(thisNode,linkNode,lk);
          // get ks from the link
          k1 = lk.k1;
          k3 = lk.k3;
          // calc magnitude of force
          forceMag = -1.0*k1*ext +
k3*ext**3;
```

Of course you will have to write the function findLink which iterates over **linkArray** and finds the link which connects **thisNode** and **linkNode**. Here's a template

```
function findLink(MAS14P_Node nodeP, MAS14P_Node nodeQ, out MAS14P_Link link) {

  local MAS14P_Link lk;
  local int i;

  for(i=0; i<linkArray.length;i++) {


  }
}
```

You will also need to clean up the **MAS14P_Link** class:

1) Declare **k1** and **k3**;

2) Change **SetParam(…)** so it can set **k1** and also **k3**.

Finally back to **MAS14P_TrussConfig.uc** You will have to set link.k1 = k1 etc in the function **spawnAllLinks(…)**

Adding code to log one node over time

1) Declare the iterator variable up top **var int itn;**

2) Create a function to log the data

```
function logData() {
  local int m;
  m = int(logInterval/deltaT);
  if( itn % m == 0) logDataRecord();
}
```

2) make a call to **logData()** inside **computeTimer()**

3) Add these lines to **Actor_Initialize(…)**

```
  openMatlabFile();
  setLogFileColumnLabels();
```

4) Add this function somewhere. You will need to decide which variables you want to log.

```
function setLogFileColumnLabels() {
  columnLabels.length = 0; //empty array.
  columnLabels[0] = "time";
  // Add more labels here

  writeMatlabFileHeaderNew(columnLabels);
}

function logDataRecord() {
  local array<float> dataArray;
  dataArray.length = 0; //empty array.
  dataArray[0] = time;
  // add more here – should match up with your
labels
```

```
  writeMatlabFileRecordNew(dataArray);

}
```

5) Add the following if statement to the function **ProcessKey(…)**

```
if(Key == 'F12'){
  bRunning = false;
  closeMatlabFile();
}
```

When you are running, to write the log file out to disk, then cross-hair the brick tower then hit F12. You should see the simulation stop.