School of Computing **COMP5700M Systems Programming, 2015**

Coursework Assignment 1: C programming

Issued: 5 October 2015 10:00
Due: 19 October 2015 10:00
Weighting: 20% of module assessment.
Typical Workload: 20 hours.

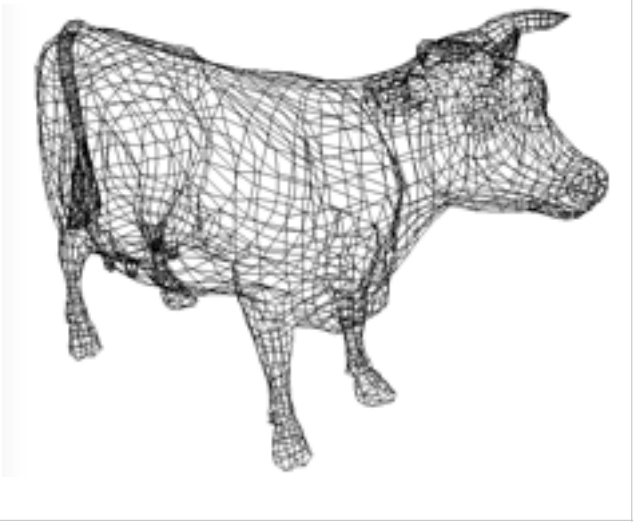
Background: This coursework is intended to develop your C programming. Although your are welcome to develop your program on your own machine, your final submission must be compilable and runnable on the School's linux machines.

Resources: The directory /home/csunix/scsdjd/Public/SysProg/ contains a number of files that you will need to do this coursework:

cow.obj	a sample input OBJ file.
redcow.obj	a sample output OBJ file.
obj.py	a Python script for visualizing an OBJ file.

Overview

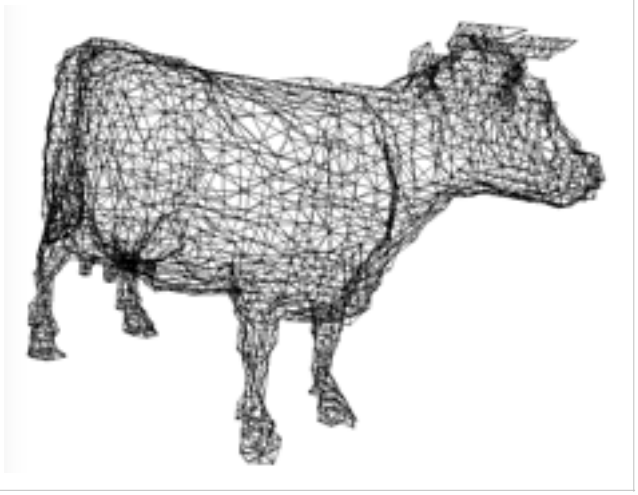
Geometric meshes arise in a variety of applications, including computer graphics and games, and in the visualisation of 3D structures. A simple geometric mesh consists of a set of vertices in 3D space, and a set of triangles built from the vertices. The figure below shows an examples of a 3D polygonal mesh, as a shaded surface and as a wireframe.



High-resolution geometric meshes can quickly become large. This slows down rendering and reduces frame-rates even on high-performance machines, and the storage space is also an issue for games where the hardware platform may have much tighter resource constraints. Mesh simplification techniques can help: they transform a mesh into one with fewer vertices and triangles, usually with some loss of precision. For many applications, the errors introduced by simplification are tolerable, and the benefits of a smaller mesh outweigh other considerations.

In this coursework you are going to implement a simple mesh simplifier: a program that will read a geometric mesh stored on disk in the OBJ format, simplify it, and write out a new OBJ file with the simplified structure. The task is set out as a series of steps, you will earn

some credit for each step undertaken even if you do not complete the full program. Marks will be awarded for correctness of your code, for appropriate use of C language features, and for the measurement and evaluation of your program's performance. The figure below shows the same mesh once is has been reduced in resolution using the algorithm you are going to implement.



For information, the input mesh has 2903 vertices and 5804 faces; the output in this case, on a mesh of resolution 80, has 2245 vertices and 4618 triangles.

Task 1. Reading an OBJ file

OBJ files contain an ASCII encoding of information about a 3D geometry model. They support a range of primitives relevant to 3D modelling and graphics, but only two are of interest for this coursework: vertices in 3D space, and triangular faces specified in terms of vertices. In a file, a vertex is specified by a line beginning with "`v`", followed by three floating point values representing the position of the vertex in a 3D coordinate system defined by x, y, and z axes. For example,

```
v 1.5 2.1 -4.4
```

is the vertex at x coordinate 1.5, y coordinate 2.1, and z coordinate -4.4. Note that there is no vertex number. Instead, vertices are numbered implicitly starting from 1. I.e. the first vertex to appear in the file is vertex 1, the second is vertex 2, etc.

A triangular face is defined by a line beginning with "`f`", followed by information about the three vertices that make up the face. For example

```
f 4 1 3
```

specifies a triangle made from vertices 4, 1, and 3. However, OBJ face information can include other vertex parameters that are not of concern to us here; for example the following are also valid specifications for the same triangle:

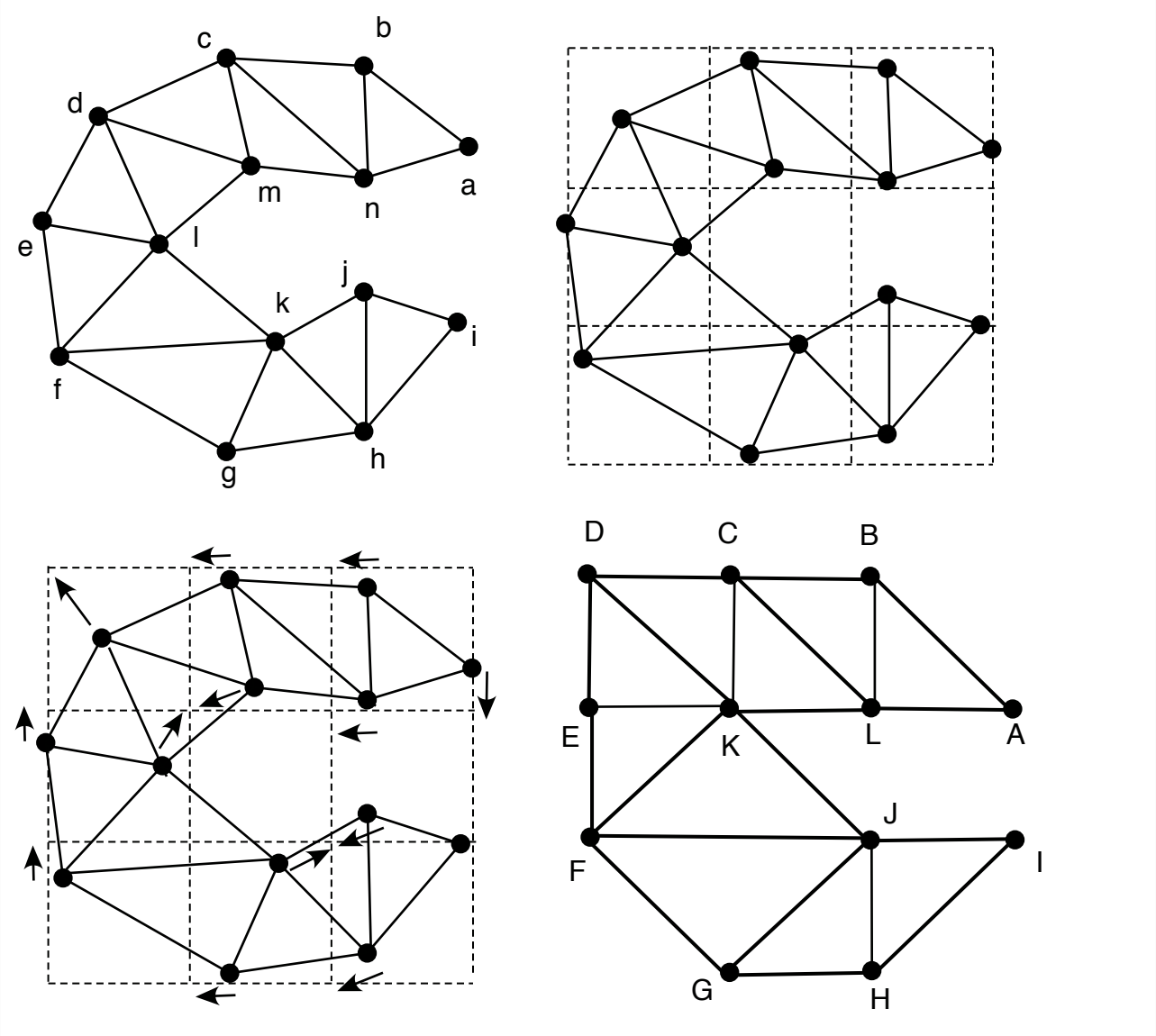
```
f 4/6 f 1/7 3/9
f 4//7 1//2 3//11
f 4/6/7 1/7/2 3/9/11
```

No matter how the face vertices are specified, we are only interested in the first number of each "group", which is the vertex number. The groups of numbers defining vertices are separated from each other (and the "`f`") by a single space character.

The name of the input and output files should be specified as command-line arguments to your program (i.e. using argv).

Task 2. Simplify the geometry.

The simplification algorithm you will implement is straightforward but rather naive; it is illustrated here using a 2D example.



The top-left of the figure shows a 2D triangular mesh. We start by computing the bounds of the mesh, i.e. enclose the shape in a 2D box, and then subdivide this box into a pre-determined number of rows and columns. Note that the bounding box size has equal sizes (i.e. its a square), and is defined so that it touches the figure on its widest dimension (here the horizontal dimension). I've chosen a new resolution of 4 x 4, and the new mesh is shown using dashed lines. We now map each vertex in the original mesh to a vertex in the new mesh. Not all vertices in the new mesh will be used, and (hopefully) multiple vertices in the input mesh will map to each vertex in the new mesh (otherwise we wouldn't get simplification). Here for example input vertices l & m are both mapped to K in the output, and j & k both map to J in the output.

For each triangle in the input mesh, we determine a new triangle in the output mesh by looking at where the vertices of the input are mapped to. Consider triangle a-b-n in the input: the vertices are mapped as follows: a->A, b->B, and n->L. So the corresponding

output triangle is A-B-L. Sometimes two or three vertices of an input triangle will map to the same vertex. As the result isn't a well-defined triangle, that input triangle is omitted from the output: input triangle d-l-m maps to D-K-K, so doesn't appear as a triangle in the output. It is also possible (though there is no example here) that two input triangles map to the same output, in this case only one copy of the triangle should be retained.

In this example, the original mesh of 14 vertices and 12 triangles has been "reduced" to an output mesh of 12 vertices and 10 triangles. Clearly the fewer vertices and meshes in the output, the coarser the approximation to the input. *In practice, mesh simplification algorithms are considerably more complicated and take into account the error between input and possible output meshes.*

Your task is to do the same thing with a 3D mesh. The principles are the same, except now you have an additional dimension in which the points are embedded. The mapping of old vertices to new is straightforward: use division and rounding! The required resolution of the output mesh should be set as a constant in the file. Your bounding volume should be a cube, and you should embed the mesh into the cube so that the longest dimension of the mesh defines the side length of the cube.

Your program will need to use a data structure to implement the check to avoid repeated triangles in the output, and you will be awarded marks for this implementation.

Task 3. Output a new OBJ file.

You should write your compressed mesh out to a new OBJ legal file. Your implementation should respect the requirement that the output should contain only well-defined triangles, and should not contain any repeated triangles. Bear in mind the comment earlier that vertices in OBJ files are implicitly numbered from 1.

Deliverable: Your submission should be a unix tar file (not zip, not rar - only tar will be accepted) consisting of one or more .c and .h files containing your implementation. This should be uploaded to the VLE into the Coursework 1 submission area. Your code must be compilable using gcc on the School's linux system. Instructions for compilation should be included in the header of your main file. Use of C++ or any other language is not permitted.

Code that does <u>not</u> compile will be marked, but you will incur a 10-mark reduction.

Marking Scheme:

The coursework will be marked out of 50, as follows:

1. Correctness: 20 marks. 5 marks for reading the data from an input file, 10 marks for computing a compressed mesh, and 5 marks for writing the output.
2. Data structure used to check for duplicate triangles: 10 marks.
3. Structuring, layout and commenting: 10 marks.
4. Use of C language features: 10 marks.

As a rough guide, you should be able to implement the program in 2-300 lines of C.

To help you I have provided a Python script that uses the VTK library to read and render an OBJ file. To use this, copy the file "`obj.py`" to your directory and issue the command

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/csunix/scsdjd/
Public/lib/

export PATH=$PATH:/home/csunix/scsdjd/Public/bin/

vtkpython obj.py <path to OBJ file>
```

Your submission must be individual work, and your attention is drawn to the University's rules regarding plagiarism. Any suspected cases of plagiarism will be investigated, and if confirmed could result in serious penalties.

Questions: Any question about this coursework should be sent by email, to D.J.Duke@leeds.ac.uk. I will reply individually to all questions; any information to be broadcast to the class will again be sent via email.