

**SCHOOL OF COMPUTER SCIENCE  
COURSEWORK ASSESSMENT PROFORMA****MODULE & LECTURER:** CM0304 Graphics, F. C. Langbein**DATE SET:** 22<sup>nd</sup> October 2007**SUBMISSION DATE:** 3<sup>rd</sup> December 2007**SUBMISSION ARRANGEMENTS:**

Submit the source code, a brief description, and an executable in a zip, tar.gz, tar.bz2 or rar archive via the blackboard assignment module by 17:00. If there are problems with this contact the lecturer by e-mail at F.C.Langbein@cs.cf.ac.uk.

**TITLE:** Graphics Coursework

This coursework is worth 25% of the total marks available for this module. The penalty for late or non-submission is an award of zero marks. You are reminded of the need to comply with Cardiff University's Student Guide to Academic Integrity. Your work should be submitted using the official Coursework Submission Cover sheet.

**INSTRUCTIONS**

Complete the OpenGL programming task as described in the attachment. Submit

- the *source code* with clear, concise comments;
- a *plain text* file 'README' briefly describing how to compile and run the program on which platform and which advanced features were implemented;
- an *executable* running on one of the platforms available at the School of Computer Science.

You may use any suitable programming language on any platform with OpenGL support available at the Cardiff School of Computer Science. Your program may make use of any core libraries of your chosen programming language (standard C/C++ library, standard maths library, STL for C/C++; core API for Java). But you may only use the standard OpenGL libraries for graphics programming (GL, GLU, GLUT with platform dependent windows interface for C/C++; the JOGL API and any Java version of the GLUT code) and any other specialised libraries mentioned *explicitly* in the attachment.

You may reproduce small code fragments from

- the CM0304 handouts, notes, web-site, tutorials and lab classes
- any recommended and background textbooks for CM0304 as available from the printed text, CDs/DVDs, websites, etc. for the textbook

provided that the sources contain clear reference to the origin of the code. You may use these code fragments to setup and initialise OpenGL, the windows environment, obtain example models, etc. But you have to write your own code to implement any of the features required for the tasks described in the attachment.

You may *not* reproduce code written by any other student or code obtained from any other source not mentioned above. If you are in doubt about whether you may include a code fragment that you have not written yourself, ask the lecturer.

**CRITERIA FOR ASSESSMENT**

Credit will be awarded against the following criteria.

	Excellent	Good	Adequate	Poor
<b>Functionality:</b> to what extent does the program realise the task described? 40%	efficient and complete implementation; all special cases are considered	feasible implementation, but not optimal; not all special cases are considered	progress towards a full implementation, but not fully working with major deficiencies	little or no progress towards implementation; approach not suitable
<b>Design and Structure:</b> how clear is the structure of the code and how well are data structures and algorithms used? 40%	well structured code with highly suitable data structures and elegant, clear algorithms	good structure with suitable data structures and algorithms; sometimes not optimal	attempt of using appropriate data structures and algorithms visible; sometimes structure is confusing	code is mostly obscure; little or no structure is visible in use of data structures and algorithm design
<b>Code Documentation:</b> how easy is it to understand the code with the comments provided? 20%	clear, concise comments describing ideas and high-level structure without unnecessary detail	clear comments about high-level structure and ideas; sometimes incomplete or too focused on details	some comments about the structure and ideas present, but hard to follow and too focused on details	hardly any comments or only very confusing or low-level comments about single instructions

Feedback on your performance will address each of these criteria.

**FURTHER DETAILS**

Feedback on your coursework will address the above criteria and will be returned in approximately 3 weeks. This will be supplemented with oral feedback in lectures and tutorials. If you have any questions relating to your individual solutions talk to the lecturer or the tutor.

## CM0304 Graphics Coursework (2007)

Write a program using OpenGL to render a 3D scene, which can be navigated in a walk-through fashion. You are free to choose the scene, but the scene and your program should exhibit the features described below. The associated marks are listed in brackets. Emphasis is placed upon the implementation of the functionality as described below using OpenGL, the appropriate use of data structures and algorithms for the design and structure of your program, and clear and concise comments in the code describing its high-level structure and the ideas used for the implementation (see Criteria for Assessment). Add comments to the source code clearly indicating which part of the source code implements which feature listed below by referring to the labels such as 1(a), 1(b), 2(a), . . . Note that you have to submit a working program, which is compilable from the sources you provide and uses only OpenGL to render the scene.

### 1. Rendering Framework [5]

Implement a framework to render a small shaded 3D scene with OpenGL which properly initialises the OpenGL pipeline, the windows environment, light sources, etc. It should provide the following functionality:

- (a) A **rendering** system which sets up the scene using display lists where possible and continuously renders the scene. It is important that the rendering system is clearly structured to easily add and remove objects, etc. from the scene (see below for the object types).
- (b) A **navigation** system which allows the user to display the scene from an arbitrary camera position with keyboard and/or mouse control to facilitate moving about the scene by adjusting the view direction and moving forward/backward (or a similar 3D navigation system). This should change the view in a natural and predictable manner.

### 2. 3D Objects [10]

Use the rendering framework to create a scene which contains shaded objects of the types described below which can all be represented as polygonal meshes. Your scene should contain at least one of each object type. The marks are on the way the types are realised, not the amount of objects.

- (a) **Parametric surface:** a curved surface described by a parameter function which is *not* a plane, sphere, cylinder or cone. The approximation accuracy of the surface by polygons should be adjustable by a parameter.
- (b) **Polyhedron:** a closed (water-tight) object that is bound only by planar surfaces, e.g. a cube. There should be at least one polyhedron in your scene and the rendering function should be able to draw arbitrary polyhedra specified by certain parameters or data structures.
- (c) **Stanford bunny mesh:** the Stanford bunny mesh as available from <http://www.langbein.org/files/graphics/bunny3.ply> (or blackboard) in ASCII PLY file format (see <http://graphics.stanford.edu/data/3Dscanrep/> or [http://www.cc.gatech.edu/projects/large\\_models/ply.html](http://www.cc.gatech.edu/projects/large_models/ply.html) for details about PLY; you may use the PLY tools to read the mesh, but it may be simpler to parse the file yourself under the assumption that all facets are triangles and you only read the  $x$ ,  $y$ ,  $z$  coordinates). Note that you have to compute *suitable* normals for the mesh yourself! (To save bandwidth do not submit the bunny mesh with your sources, but assume that it is available in the current directory).

### 3. Advanced Features [10]

Implement *two* of the following advanced features. Clearly indicate in the sources which one of these you have chosen, you will only get marks for two of them.

- (a) Create an object which consists of at least three **transparent** polygons. Note that when you render more than one transparent polygon you have to make sure that there are no visible artefacts from every camera position. Your code has to take care of this explicitly and you should not arrange the transparent polygons such that no artefacts are visible.
- (b) Implement a **particle effect** such as steam from a teapot or smoke from a fire or bubbles in water. Try to make it look as realistic as possible by, e.g., using polygons and textures and not just separate points.
- (c) Smooth the Stanford bunny mesh by applying a suitable **subdivision** scheme multiple times to it before you render the mesh.
- (d) Impress us with **something else**. . . . But ask the lecturer whether your idea is suitable and clearly state in your submission what you are attempting to do.

If you have any questions about this coursework, you are welcome to contact the lecturer.