CM20219 – Coursework Part 2 – 2020/2021

Viewing and analysing 3D Models using WebGL

Dr. Wenbin Li

1. Introduction

In this coursework, you will implement software to render and manipulate a 3D model using WebGL. This assignment is worth 20% of the total marks for the unit.

2. Deliverables and assessment

You will write software using JavaScript and WebGL. You will be given a skeleton WebGL application based on Three.js (a widely-used JavaScript graphics API) to start with, and should attempt to implement each feature described in the requirements table given later in this document.

This part of the coursework has two deliverables.

The first is the **code** submitted on Moodle. We will run your code and mark it against several categories, most of which will be based on results alone. **Your code will also be checked to ensure you are obeying good practices.** You should submit a **README** file to indicate how to reproduce the tasks.

For the second deliverable, you will submit a report documenting the **entire** coursework (not the PYTHON part). It is important that you thoroughly describe the mathematics that you have used, and how you implemented this using JavaScript and Three.js (including how you overcame any difficulties).

- The report should demonstrate your understanding of the concepts and show how you implemented key elements (with short, relevant sections of source code included in the flow of the text). Your report should include some evidence of testing, including screenshots and numerical results where appropriate. It is important to highlight the capabilities and limitations of your software, and clearly describe how you could improve it if you had additional time. **The page limit is 8 pages.**

- Source code should be easily readable by a programmer. This means appropriate use of variable names and comments.

Please organise your report by requirement number using the table given later in this document.

You must submit your source code and an electronic copy of the report (PDF) via Moodle.

Please note the deadlines for the deliverables for this part of the coursework:

Code (50%) + Report (50%): Friday 11th December 2020, 20:00 via Moodle

Feedbacks will be given via Moodle in two weeks after the deadline.

3. Getting started

You will write this part of the coursework in JavaScript, using Three.js which wraps WebGL. All computers in the university should be able to run WebGL in Edge, Chrome or Firefox. WebGL should also work on your own computer and even phone or tablet. The tutors will be able to assist you with getting started, as well as answering questions about JavaScript, Three.js and graphics concepts.

You can download a skeleton framework from Moodle, which is set up to show you a minimal example to get started straight away. You can use any text editor you like for editing files, although it might be convenient to use Visual Studio as it provides syntax highlighting and other helpful development tools. After saving your files, refresh the browser tab to update it.

To get started, look through the code in "WebGLCoursework.html" and make sure you understand how Three.js is initialised, and how the "`animate`" and "`handleResize`" callback functions are registered and implemented. Then you should look at how the scene is being drawn, to start working on your requirements. The following short presentation is also a good introduction to Three.js (by Eric Haines): http://www.realtimerendering.com/basics3js/#1.

You can find the full Three.js documentation at:
https://threejs.org/docs/index.html#manual/introduction/Creating-a-scene

As soon as you want to load images for textures or meshes from the local drive, you will see that the security settings of internet browsers will block it. Three.js has a page on how to run things locally: https://threejs.org/docs/#manual/introduction/How-to-run-thing-locally. Perhaps the simplest workaround is to start a local webserver and access your coursework via it:

1. Install PYTHON from: https://www.microsoft.com/en-gb/p/python-37/9nj46sx7x90p
2. Open a command line (or terminal on Linux or macOS) in the directory where you have your coursework. (On Windows, you can press shift + right click, and then select "Open command window here").
3. Start a simple webserver with Python. On a university computer, you can type: "`python -m http.server PORT`"
   You will tell the system at which port the server will be listening (e.g. 32007).
4. In your web browser, navigate to "`http://localhost:PORT`", where "PORT" is the number you used in Step 3.

4. Learning outcomes

After completing this coursework, and attending the relevant lectures, you should be able to:

- Construct appropriate mathematical operations to manipulate 3D objects
- Use an industry standard graphics programming API (WebGL)
- Develop code to interact with JavaScript events (mouse clicks, key presses etc.)
- Test and document a graphics interaction system

5. Plagiarism

You must complete this coursework individually. If you copy code from another student and include it in your project without clear attribution, then you have committed plagiarism. Plagiarism is a serious academic offence. For details on plagiarism and how to avoid it, please visit http://www.bath.ac.uk/library/help/infoguides/plagiarism.html. Undetected plagiarism degrades the quality of your degree, as it interferes with our ability to assess you and prevents you learning through properly attempting the coursework. Consequently, if we detect any plagiarism you will receive zero marks for the coursework and be referred to the Director of Studies for disciplinary action. Such action may affect your ability to continue your studies at the University. Note that properly attributed code, while allowed in your submission, will not contribute towards your marks. The report marks will also only be applicable to sections you have coded yourself.

6. Coursework requirements specification

This coursework will be marked based on your code (50%) and your report (50%).

The following table details the required features to implement, as well as the marks available for each requirement. Please see the detailed marking scheme in the following section for a breakdown of marks by requirement.

Req#	Feature to implement	Weighting
1	**Draw a simple cube** The cube should be centred at the origin (0, 0, 0), with opposite corner points at (–1, –1, –1) and (1, 1, 1), with its faces orthogonal to x-, y-, z-axes.	10%
2	**Draw coordinate system axes** Draw colour lines to represent the axes (use RGB for XYZ) – no arrows or tick marks necessary.	10%
3	**Rotate the cube** Rotate about the x-, y-, and z-axis, respectively (axes and camera should not move).	10%
4	**Different render modes** Implement a keyboard shortcut for rendering the cube using only 3 rendering modes: vertices, edges, faces.	10%
5	**Translate the camera** Manipulate the camera's location by translating it along its up/down, left/right, and forward/backward directions.	10%
6	**Orbit the camera** Manipulate the camera by rotating it about the look-at point ("arc ball" mode).	10%
7	**Texture mapping** Map different textures on different faces of the cube.	10%
8	**Load a mesh model from .obj** Load the mesh (e.g. the Stanford bunny), and then translate and scale it uniformly to fit into the cube	10%
9	**Rotate the mesh, render it in different modes** as you did for the cube	10%
10	**Be creative – do something cool!** Some ideas include: using shaders, animations, shadows, reflections, picking and moving an object, contribute to the Three.js documentation to improve it etc.	10%

7. Demo marking scheme

For each task, the description mentions the functionality that is expected for full marks (100%). Partial implementations lead to deductions, and missing features are marked as 0%.

Task	Weight	Description
1. Draw a simple cube [10%]		
a	60%	A cube is drawn onto the screen.
b	40%	Cube is centred at the origin (0, 0, 0), with opposite corner points (–1, –1, –1) and (1, 1, 1), faces orthogonal to x-, y-, z-axes (check code).
2. Draw coordinate system axes [10%]		
a	50%	Three orthogonal lines are drawn to represent the x-, y- and z-axes of the world coordinate system.
b	50%	The axes are drawn in red (x), green (y) and blue (z), respectively.
3. Rotate the cube [10%]		
a	1 / 3	Rotation of the cube about the x-axis (with axes + camera fixed). (Remark: it might help to reset rotation between tasks.)
b	1 / 3	Rotation of the cube about the y-axis (with axes + camera fixed).
c	1 / 3	Rotation of the cube about the z-axis (with axes + camera fixed).
4. Different render modes [10%]		
a	1 / 3	Vertex render mode shows the 8 vertices of the cube.
b	1 / 3	Edge render mode shows edges of primitives. This is to draw the cube’s mesh using only edges, in what is known as ‘wireframe’ mode.
c	1 / 3	Face render mode shows the 6 faces of the cube (obviously only at most three faces at any one time).
5. Translate the camera [10%]		
a	1 / 3	Translate the camera along the camera's left/right vectors, not the axes of the global coordinate system.
a	1 / 3	Translate the camera along the camera's up/down vectors, not the axes of the global coordinate system.
c	1 / 3	Translate the camera along the camera's forward/backward vectors, not the axes of the global coordinate system.

Task	Weight	Description

6. Orbit the camera [10%]

a	70%	Orbit the camera about the cube (so-called “arc ball” mode), i.e. move about the look-at point at a fixed distance. The look-at point can be flexibly changed to any location within the scene. In fact, you are required to deliver the implementation similar to three.js’s OrbitControl function where the look-at point can be moved, and you **CANNOT** use the three.js’s build-in OrbitControl function for this purpose.
b	30%	The camera can orbit in both latitude and longitude directions.

7. Texture mapping [10%]

a	40%	The cube has a texture applied to it.
b	30%	Correct texturing (without any skew) and perspective rendering.
c	30%	Each face should look different.

8. Load a mesh model from .obj [10%]

a	50%	Load and display a mesh model (e.g. the Stanford bunny).
b	50%	Correctly (uniformly) scaled and translated to fit inside the cube.

9. Rotate the mesh, render it in different modes [10%]

a	20%	For the loaded model, rotate it about the x-, y- or z-axis.
b	20%	For the loaded model, show the vertex rendering mode.
c	20%	For the loaded model, show the edge rendering mode.
d	40%	For the loaded model, show the face rendering mode (with materials, lighting and shading).

10. Be creative – do something cool! [10%]

a	100%	Go beyond the previous tasks. Using shaders for creating new materials, load and animate meshes (beyond rotations), add a ground plane that your objects through shadows on, draw multiple objects (e.g. a solar system with orbiting planets) … anything really.

8. Report marking scheme

The report (max. 8 pages) should be organised by requirement number and make clear what has been implemented. See Section 2 for more details.

Explanations – per requirement [20%]

– tasks to perform and how they're achieved
– description of mathematical background

Implementation – per requirement [20%]

– relevant code snippets
– discussion and explanation of relevant functions

Form / Organisation [15%]

– structure of the report (by requirement number)
– clarity of exposition, language
– appropriate formatting (page numbers, code snippets as text, with syntax highlighting)
– using appropriate illustrations (with correct attribution and informative captions)
– referencing (to figures, sections etc.) and references

Evaluation / Testing – per requirement [10%]

– show that it works (no need to replicate every single part of the demo)
– use of screenshots or numerical results as appropriate

Discussion [5%]

– Lessons learned
– Limitations
– Future work

Code quality [5%]

– clear code structure
– code clarity (e.g. appropriate variable names, functions, comments)
– using appropriate functions from libraries

Extensions [25%]

going beyond the coursework requirements in functionality, documentation, testing, discussion, code quality, report typesetting etc.

Penalties

50% – no (own) code submitted
5–20% – text or code plagiarism (depending on severity)
5% – report not submitted as a separate document
5% – incomplete code submission, e.g. missing images or scripts
Note: these penalties are given in percentage points.