

Your mission in Project A is to use WebGL and HTML-5 to:

- a) Draw several moving, turning, jointed colored shapes with OpenGL's basic drawing primitives (various forms of points, lines and triangles, etc.) using vertex buffer objects full of 3D vertex attributes.
- b) Use a modelMatrix-like matrix stack to transform those shapes them interactively (see 'stretched robot' code)
- c) Ensure that interesting parts of your on-screen image move continuously without user input (animation) and
- d) Make some parts of at least one jointed object move smoothly in response to keyboard and mouse inputs.

Our goal is to learn enough WebGL to make an *interactive* drawing that **you find interesting and compelling.** You may choose to draw ANYTHING with multiple joints: an octopus? Fractal trees that grow and wave in the wind? An N-legged walking creature whose legs consist of 2 or more segments? (Google/Bing: 'DaintyWalker', 'Strandbeest', etc.) A human-like hand that opens and closes (Google/Bing: 'early 3D graphics' 'Catmull hand video', 'Red's Dream')? A car? Bicycle? Solar system with moons? bird? helicopter? spaceship?

Requirements:**Project Demo Day (and due date): Mon Jan 25, 2016**

A)-- In-Class Demo: on the Project's due date (Mon Jan 25) you will demonstrate your completed program to the class. Two other students will each evaluate your work on a 'Grading Sheet', as may Professor Tumblin and assistants. Based on Demo Day advice, you then have ≥ 72 hours to revise and improve your project before submitting the final version for grading. Your grade will mix Demo Day grading sheets + your improvements.

B) -- Submit your finalized project to CMS/Canvas no more than 72 hours later (11:59PM Thurs Jan 28) to avoid late penalties. Submit just one single compressed folder (ZIP file) that contains:

- 1) your written project report as a PDF file, and
- 2) one folder that holds sub-folders with all Javascript source code, libraries, HTML, etc. (mimic the 'starter code' ZIP-file organization) We must be able to read your report & run your program in the Chrome browser by simply uncompressing your ZIP file, and double-clicking an HTML file found inside, in the same directory as your project report.

---IMPORTANT: Name your ZIP file and the directory inside as: **FamilynamePersonalname_ProjA**

For example, my project A file would be: TumblinJack_ProjA.zip. It would contain sub-directories such as 'lib' and files such as TumblinJack_ProjA.pdf (a report), TumblinJack_ProjA.html, TumblinJack_ProjA.js ,etc.

---To submit your work, upload your ZIP file to Canvas→Assignments. DO NOT e-mailed projects (deleted!).

---BEWARE! **SEVERE LATE PENALTIES!** (see Canvas→Assignments, or the Syllabus/Schedule).

Project A consists of:

1)—Report: A short written, illustrated report, submitted as a printable PDF file.

Length: >1 page, and typically <5 pages, but you should decide how much is sufficient.

A complete report consists of these 3 sections:

- a)--your name, netID (3 letters, 3 digits: my netID is jet861), and a descriptive title for your project (e.g. Project A: Planetary Gear Transmission, not just Project A)
- b)—a brief 'User's Guide'. Begin with a paragraph that explains your goals, then give user instructions on how to run and control the project. (e.g. "A,a,F,f keys rotate outer ring forwards/backwards; S,s,D,d keys rotate inner ring forwards/backwards; HUD text shows velocity in kilometers/hour.") Your classmates should be able to read ONLY this report and easily run and understand your project without your help.
- c)—a brief, illustrated 'Results' section that shows **at least 4 still pictures** of your program in action (use screen captures; no need for video capture), with figure captions and text explanations.

2)—Your Complete WebGL Program, which must include:

- a) **User Instructions:** **When your program runs and/or when the user presses the F1 (help) key**, print a brief set of user instructions onscreen somewhere. How? You decide! Perhaps put them in the webpage, outside

of the ‘canvas’ element where WebGL draws pictures, or within the ‘canvas’ element using the ‘HUD’ method in the book, or in the JavaScript ‘console’ window (in Google ‘Chrome’ browser), etc. Your program must not puzzle its users, or require your presence to explain, find, or use any of its features.

b)— At least two different student-designed 3D ‘parts’. Your program must use WebGL calls to draw at least two different and colorful 3D parts on-screen; more complex than a rectangle or a cube (at least one with more than 12 vertices), made by one or more calls to draw the contents of a Vertex Buffer Object (VBO). (In step d) below, you will assemble these parts to make moving, jointed objects).

c)— Per-Vertex Colors. Each vertex of each part must have its own individually-specified color attribute (see ‘multiple attributes’ in your book), to cause on-screen interpolation of vertex colors within each WebGL drawing primitive. (Do not substitute ‘canvas’ drawing primitives (e.g. `context.filledRect()`); use vertex buffer objects (VBOs) for sets of vertices as demonstrated in WebGL Programming Guide, Chapter 3,4).

d)—At least two (2) or more Different Kinds of jointed, moving objects assembled from ‘parts’. Your project must demonstrate how trees of transformations can make jointed objects from parts drawn in nested ‘drawing axes’ or ‘reference frames’ in WebGL. You must construct and draw at least two different *kinds* jointed objects, **each with at least 3 sequentially-connected hinged parts**.

For example, a one-legged hopping robot with 3 parts has a ‘torso’ part with a ‘hip’ joint to connect to a ‘thigh’ part, with a ‘knee’ joint to connect to a ‘shin’ part. You must ‘push’ and ‘pop’ matrices from your `glModelView`-like matrix stack to enable drawing these parts connected by joints. Each and every joint in your objects must rotate smoothly (no ‘jumps’), yet stay pinned together as joints move, as if connected by hinges.

Your program and its shaders must create `ModelMatrix`-like concatenations of 4x4 matrices to transform all the vertices of your object, to position, scale, and orient/rotate them in pleasing ways. Construct your ‘model’ matrix in Javascript using the `cuon-matrix-quat.js` library supplied in the starter code, then apply it to the contents of vertex-buffer objects (VBOs) in your Vertex Shader.

Warning! I will not accept substitutes for matrix usage, such as “`x = x + xtrans; y = y + ytrans;`”

e)--Animation! Like all projects in this course, your program must show a picture that moves and changes, both by itself (animation) and in response to user inputs (interaction) from mouse or keyboard. Users must be able to pause/unpause the animation, and to move objects to anywhere desired on the HTML-5 canvas.

d)—Smooth movement only: As your objects move due to animation and/or user inputs they must travel smoothly, continuously; they must not make any sudden ‘jumps’ from one position to another.

c)—Event Handlers: your program and its shaders should make proper use of registered event handlers for **keyboard, mouse and display**. You have many choices here, including the simple methods demonstrated in Chapter 3 and in the ‘starter code’ posted. Event handlers let your programs respond to the mouse, respond to changes in the display window size, respond to keyboard inputs, and more. You are also welcome to use external libraries for user-interfaces in HTML/Javascript, such as basic CSS controls or Google’s `dat.GUI`: <http://workshop.chromeexperiments.com/examples/gui/#1--Basic-Usage> ; <https://code.google.com/p/dat-gui/>

3)—Note all the opportunities for extra credit by adding more features to your project; see Grading Sheet.

Sources & Plagiarism Rules:

Simple: never submit the work of others as your own. You are welcome to begin with the book’s example code and the ‘starter code’ I supply; you can keep or modify any of it as you wish without citing its source. I strongly encourage you to always start with a basic graphics program (hence ‘starter code’) that already works correctly, and incrementally improve it; test, correct, and save a new version at each step. Also, please learn from websites, tutorials and friends anywhere (e.g. `.gitHub`, `openGL.org`, etc), please share what you find on Canvas Discussion board (but NEVER post code you will turn in for grading—only ideas or examples), but you must always properly credit the works of others in your graded work—**no plagiarism!**

Plagiarism rules for writing essays apply equally well to writing software. You would never cut-and-paste paragraphs or whole sentences written by others and submit it as your own writing: and the same is true for whole functions, blocks and statements. Never try to disguise it by rearrangement and renaming (TurnItIn won’t be fooled). Instead, study good code to grasp its best ideas, learn them, and make your own version in your own style. Take the ideas alone, not the code: make sure your comments properly name your sources. (And, Ugh, if I find plagiarism evidence, the University requires me to report it to the Dean of Students for investigation).