EECS 351-1: Intro to Computer Graphics **Project A: Moving Shapes**

Instructor: Jack Tumblin **Winter 2015**

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 glModelView-like matrix stack to transform those shapes them interactively,
c) Ensure that 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 Feb 2, 2015**

A)-- In-Class Demo: on the Project's due date (Mon Feb 2) you will demonstrate your completed program to the class. Two other students will each evaluate your work on a 'Grading Sheet', as will 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 combines grading sheet scores & improvements.

B) -- Submit your finalized project to CMS/Canvas no more than 72 hours later (11:59PM Thurs Feb 5) 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) just one sub-folder that holds all your Javascript source code, libraries, HTML, and any support files you made: we must be able to read your report run your program in the Chrome browser with nothing more than the files you sent us.
---IMPORTANT: Name your ZIP file and the directory inside as: **Familyname**Personalname_ProjA
For example, my project A file would be: TumblinJack_ProjA.zip, and it would contain my report file namedTumblinJack_ProjA.pdf, TumblinJack_ProjA.html, TumblinJack_ProjA.js ,etc,
---To submit your work, use Canvas→Assignments. DO NOT e-mail your project!
---BEWARE! SEVERE LATE PENALTIES! (see Canvas→Syllabus→'Course Details')

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, 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; arrow UP/DN keys adjust the electric motor speed, left/right keys act as the car's accelerator, 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 starts 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 understand and use all 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) you assemble these parts into 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 might have body, thigh, and shin parts. You’ll need to ‘push’ and ‘pop’ matrices from your glModelView-like matrix stack as you draw some of the parts. Each and every joint in your jointed objects must rotate, yet stay pinned together as if connected by hinges.

Your program and its shaders must create a glModelView-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 ‘modelView’ matrix in Javascript using the cuon-matrix.js library supplied by your book, then apply it to the contents of vertex-buffer objects (VBOs) in your Vertex Shader.
Warning! I will not accept hard-coded expressions for any transforms, 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 also responds to user inputs (interaction) from mouse or keyboard. Users must be able to pause/unpause the animation, and to move objects to any desired location in the 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 instantaneous ‘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 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 use any, all, or none of the book’s example code, to modify and submit the ‘starter code’ I supply, or to learn from websites, tutorials and friends anywhere (e.g. .gitHub, openGL.org, etc), but you must always properly credit the works of others—**no plagiarism!**
Share what you find with other students, too -- list the URLs on CMS/Canvas discussion board, *etc.* and list in the comments the sources that helped you write your code.

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. You must write your own code to earn your own grades: never cut-and-paste other’s code, and 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 then close that source. Take the ideas alone, not the code: make sure your comments properly name your sources, and then write your own, better code in your own style.

Also, please note that I use the ‘TurnItIn’ anti-plagiarism software on Canvas for this course; it compares your submission to your classmates, previous years’ submissions, and all web-available content. (If I find plagiarism evidence, the University requires me to report it to the Dean of Students for investigation).