EECS 351-1: Introduction to Computer Graphics — Project B: Hinged Views

Instructor: Jack Tumblin — **Winter 2014**

Your mission in Project B is to move from 2D to 3D drawing by making multiple side-by-side, colorful, interactive 3D perspective and orthographic drawings of objects moving objects on a 'ground plane' tied to 'world' coordinates. As with ProjectA, draw anything you like for your project, something *you* find interesting, meaningful and/or compelling, but now your shapes and the way you view them have shifted to 3D. Use any and all inspiration sources. How about some clockwork gears? An interactive NxN Rubik's cube(default is 3x3)? A steerable butterfly that flies around in 3D by flapping its wings? A fanciful ornithopter? A forest scene made of fractal/graftal/L-system trees and bushes ('trees of transformations')? A helicopter? A bird? Could you make motorcycles, bicycles, tricycles, perhaps with a pedal and chain-drive that really works?

Requirements: **Project Demo Day (and due date): Mon Feb 24, 2014**

1) In-Class Demo Day: when you will demonstrate your completed program to the class, and two other students will each evaluate it by filling out a 'Grading Sheet' for you, as will Professor Tumblin or TA Paul Olczak. You can then have 72 hours to revise and improve your project based on what you learned from Demo Day, before submitting the final version of your project. Your grade will combine grading sheet scores and your improvements.

Submit your finalized project to Blackboard/CMS no more than 72 hours later (Thurs Feb 27 11:59PM) to avoid late penalties. Submit just one single compressed folder (ZIP file) that contains:
a) your written project report as a PDF file, and
b) 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 of your ZIP file and name of the directory inside: **Familyname**Personalname_ProjB
For example, my project C file would be: TumblinJack_ProjC.zip, and it would contain my report namedTumblinJack_ProjC.pdf, and my project/source-code folder named TumblinJack_ProjC
---To submit your work, use Blackboard→Assignments. DO NOT e-mail your project!
---BEWARE! SEVERE LATE PENALTIES! (see Blackboard→Syllabus→'Course Details')

Project B 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 4 sections:

a)--your name, netID, and a descriptive title for your project
(e.g. Project B: Forest of Trees Fly-through, not just "Project B")

b)—a brief 'User's Guide' that starts with a paragraph that explains your goals, and includes instructions on how to make your on-screen animated images run/stop (animation) and change.

(*e.g.* "A,a,F,f keys to pan your view left/right; S,s,D,d to translate your viewpoint left/right; arrow UP/DN keys to climb/dive; arrow LEFT/RIGHT keys to make sweeping turns left or right, PgUp, PgDn keys to speed up/slow down; slowing down past zero lets you fly in reverse") 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 explanation of what the pictures are showing us, and how they help your project meet your goals.

2)—User Instructions: When your program starts and/or when users press the F1 (help) key, print a brief set of user instructions, either in the console window, the 'canvas' or in the HTML page that contains your canvas (e.g. 'arrow keys move skateboard, left mouse drag steers, right mouse drag aims flame-thrower').

3)---'Ground Plane' Grid: Your program must clearly depict a 'ground plane' that extends to the horizon: -- a very large, repetitious pattern of repeated lines, triangles, or any other shape that repeats to form a vast, flat, fixed 'floor' of your 3D world. You MUST position in the **x,y plane** (z=0) of your 'world-space' coordinate system. This grid should make any and all camera movements obvious on-screen, and form a reliable 'horizon line' when viewed with a perspective camera.

4)—Adjustable, 3-Jointed, 4-segment 3D Shape: Your code must show **at least one jointed 3D object** with at least four (4) sequentially-jointed segments connected by three(3) sequential joints at different locations (one MORE than req'd for project A). Let users adjust those joint angles smoothly by interactions with the mouse and/or keyboard, as in Proj. A.

5)—Additional Separate Multi-colored Shapes: 4 or more. 'Separate' means individually positioned, spatially separate, distinct, different-shaped objects. The top part of a robot and the bottom part make up just one object, as you wouldn't move them to opposite sides of the screen! They don't have to move, but they do have to be distinct and fundamentally different, unrelated, spatially-separated dissimilar objects. 'Multi-color' means an object uses at least 3 different vertex colors, and WebGL must blend between these vertex colors to make smoothly varying pixel colors.

I prohibit use of lighting, materials, surface normals and texture maps in Project B, as we're saving them for projects C. Vertex color == Screen color.

6)—Show 3D World Axes, and some 3D Model Axes: Draw one set fixed at the origin of 'world space' coordinates, such as those found in the starter code and shown in class, and at least two others to show other coordinate systems within your jointed object.

7)--Four Viewports in a Re-sizeable Webpage: Your program must depict its 3-D scene in 4 separate viewports in the window, arranged in a 2x2 grid that together fill the entire window without gaps and without distorting the images within, even after window re-sizing makes the webpage taller or wider (e.g. an 'anamorphic'combination of viewport and camera settings). Of the 4 viewports, 3 will show fixed, orthographic views of front, top, side; and the 4th shows a 3D perspective image image with mouse-controlled viewing direction & mouse and/or keyboard controlled viewing position.

8)—View Control: smoothly, independently vary 3D Camera position and its aiming direction. Both, together! Your code must enable users to explore the 3D scene via user interaction. I recommend that you use mouse dragging and arrow keys to steer and move through the scene. You may design and use your own camera-movement system, but for full credit your system must allow complete freedom of movement in 3D:

-- your camera MUST be able to move to any 3D location from any other 3D location in a straight line: do not require users to move in circles of varying radius!

-- from any 3D location, your camera MUST be able to smoothly pivot its viewing direction without any change in 3D position (if you pretend that you are the camera, you must be able to turn your head without moving your body).

-- from any orientation, your camera MUST be able to smoothly move to any other desired 3D location without changing its orientation(if you pretend that you are the camera, you must be able to move your body without turning your head).

For example: suppose I make a scene of 64 colorful cubes placed in a 4x4x4 grid on the 'ground plane', as if they formed magical floating buildings in a city filled with futuristic flying cars. Your camera must be able to show the city from the viewpoint of any of those cars as they drive in, around, and through the 3D grid of buildings; you must invent a 3D Google StreetView-like camera. If your system cannot easily position the caera to 'drive around the block'; if your system always aims the camera at the origin, or can only move the camera in a circle around the origin, it does not meet the project requirements. BIG BIG HINTS: If you use **`LookAt()`** to create your 'view' matrix, your user controls must modify BOTH the camera position (VRP or 'eye') AND the target point or 'look-at' point, and vary them independently. In class we described the 'glass-cylinder' model for camera movement that will achieve these goals.

9) Switch 3D Cameras: In the 4th viewport of your project (the other 3: top, front, side view, orthographic) your program must let users **switch back and forth between an orthographic camera and a perspective camera**, without changing the viewing position or the viewing direction. 'Different cameras' mean you must change the PROJECTION matrix; do not change MODELVIEW for this! Note that our book's 'cuon-matrix.js' library offers both orthographic and perspective camera matrices. For extra credit, make your movable camera adjustable—try an architectural 'view' camera offers adjustable asymmetric left/right, top/bottom edges.

Outside Sources: You are welcome to use any, all, or none of the book's example code, the 'starter code' I supply or others, from websites, or any other useful resource you can find, but you must credit their work properly—no plagiarism! List the URLs on CMS/Blackboard discussion board, etc. Most importantly, you must SUBSTANTIALLY MODIFY what you find--make this YOUR code, and not just a minor adjustment to the work of others.

Have fun with this!