MP2 Ideas/Hints

How do I get started?

Getting Started: Break the Problem Down

1. Index.html
2. Javascript and flight simulator “setup”
3. Flight simulator controls
4. Terrain
5. Flight simulator camera
6. Quaternion Math
7. Shaders

Look at MP1 for #1, 2, 7.
Use the math package provided for #6.
That means, focus on #3, 4, 5.

Flight Simulator Controls-1 (Initializing)

- FlightSimulator() // Think about what variables you need to define
    - Initialize terrain vars
    - Initialize camera vars
- RunFlightSimulator() // This is a lot like MP1. What is new here?
    - Set GL parameters (e.g. depth test, face culling, etc.)
    - Bootstrap the shaders
    - Initialize the terrain data
    - Initialize matrices (e.g., view, perspective)
    - Kick off the rendering loop (and draw)

Flight Simulator Controls-2 (Processing keys)

- KeyboardEventHandler(event)
    - Get and respond to button presses that steer plane – you need an event handler
    - What event was it?
        - If (event.which == 65) … // map pitch and roll commands to keys
    - Let’s say it was a “roll” command: tell the camera about it
        - MyMP2Program.myCamera.roll(theta); // Tell the camera you are doing a roll, and how
          // much to roll, and which direction
        - MyMP2Program.myCamera.updateMatrices(); // Update view, perspective matrix,
          // because you are doing a rotation

Flight Simulator Controls – 3 (Drawing)

- You did a lot of this in MP1. What is different for MP2?
- Clear the color, screen, and depth buffer (housekeeping)
- Update the GL Matrices (View and Perspective matrices – you probably calculated the new values when you processed the pitch/roll command. Now hand them to the shaders.)
- Hand off the vertex and color data to the shaders for drawing

Terrain

- generateTerrainVertices()
  - Implement terrain generation algorithm here. We gave you an algorithm in class, but you can use any terrain generation algorithm you would like.
- This is the geometry you will draw and navigate around.
  - Put this somewhere in your code where it will be convenient to do the drawing.

Flight Simulator Camera – 1 (Initialize)

- MyCamera()
  - Set aspect ratio, fieldOfView, nearBound, farBound, and call mat4.perspective
  - Create varables for tracking rotations (what do you need for a quaternion rotation?)
  - Initialize camera values
    - What do you need to define if you are going to call mat4.lookAt?
    - What to you need to define if you are going to accumulate a quaterion rotation?
    - What should you define if you are going to fly forward at a constant speed?
    - Where are you initially? Define that. To make it easy on yourself, put the plane in a nice, neutral (unrotated) location and orientation to start.
    - Where are you facing?
    - If you are initially unrotated, how would you express that in QUATERNIONS?
    - If you are initially unrotated, where might the up vector be?

FlightSimulatorCamera -2 (Initialize)

- (Initializing, cont.)
  - Call mat4.lookAt with the values you initialized
  - If you have a quaternion value, normalize it
  - Initialize your matrices (view, perspective)

flightSimulatorCamera – 3 (applying rotation)

- Your plane (camera) will have to process pitch and roll commands. What might a “yaw processor” do?
- Yaw(degree)
  - Goal is to update the cumulative quaternion rotation
  - Use quat.create() to create a ‘scratchpad’ quaternion object to work with
    - workingQuat = quat.create(); // Create a quaternion object
  - Use quat.setAxisAngle to update workingQuat with a yaw rotation (about y axis)
    - Quat.setAxisAngle(workingQuat, axis_of_yaw_rotation, rotation_in_radians);
  - Call quat.normalize to normalize resulting quaternion
  - Call quat.multiply to update the accumulated camera rotation with the latest yaw
  - You have a new resultant cumulative quaternion – call quat.normalize to normalize
  - Now update the view matrix using this updated quaternion

FlightSimulatorCamera – 4 (Update view matrix)

- updateViewMatrix()
  - Goal is to call mat4.lookAt with new values (what does mat4.lookAt need?)
  - Apply quaterion rotation to the up vector
  - Apply quaternion rotation to the lookat vector
  - Update the lookat position (if you changed the direction you are looking)
  - Did the camera position change?
  - Call mat4.look at with the new values