A. (50) The input to a z-buffer algorithm is a soup of polygons for which the screen-space vertex coordinates are known and so is the color and opacity. Now answer the following questions:
   a. (5) What is the output of the z-buffer?
   b. (5) Provide the pseudo-code for the z-buffer algorithm.
   c. (20) Consider a z buffer with a range of 0 (at the screen) to 10 (at the far clipping plane), and a 6x6 pixel image. This is how the z-buffer looks initially (to your left).
      i. Now render following polygons - all are screen coordinates.
         1. Polygon 1: Depth=4; vertex 1=(1,1); vertex 2=(1,3); vertex 3=(3,3); vertex 4=(3,1)
         2. Polygon 2: Depth=2; vertex 1=(3,0); vertex 2=(3,2); vertex 3=(5,2); vertex 4=(5,0)
         3. Polygon 3: Depth=8; vertex 1=(2,2); vertex 2=(2,5); vertex 3=(4,5); vertex 4=(4,2)
         4. Polygon 4: Depth=3; vertex 1=(2,0); vertex 2=(2,3); vertex 3=(5,3); vertex 4=(5,0)
      ii. (10) Update the z-buffer after drawing every polygon. Show four updated z-buffers and use polygon id for color.
      iii. (5) Now, let us assume polygons 3 and 4 are opaque, and the other two are transparent. Mark all pixels where a plain z-buffer will work and will not work.
      iv. (5) At pixels (3,2) and (4,3) write down the blending that could be performed (if applicable). Let the opacities for polygon 1 and 2 be 0.5 each. Assume that these
two polygons are with colors given the polygon id (the color for polygon 4 is $C_4$, etc.).

d. (10) Explain how efficiency is obtained in a scanline version of the z-buffer. What quantity is computed efficiently? Write down the complexity of the algorithm. Choose the number of pixels to be $N$ and the number of polygons that land on the Z-buffer to be $P$. Write down the steps.

e. (10) Explain the differences in the A-buffer and Z-buffer algorithms and especially mention the rationale for the existence of the A-buffer. We went over the A-buffer algorithm in class and it is in the notes.
B. (30) Clipping - Answer the following questions about clipping
   a. (5) Provide pseudo-code for the 2D Cohen-Sutherland algorithm. What are the inputs and outputs? What is the complexity of the algorithm.
   b. (5) How are full polygons handled since the Cohen-Sutherland algorithm is line- and point-based?
   c. (5) How will you change the 2D Cohen-Sutherland to work in 3D? What is the complexity of the 3D algorithm?
   d. (10) Consider the following polygons (red dotted line) and a single clip window (bold line). How will the algorithm progress for each of the polygons? Show final output. Show all steps.
C. (20) Hierarchical Models
   a. (10) Consider a three-wheeled scooter rickshaw. Each of the three wheels rotate and the rickshaw translates. Make a hierarchical model similar to the car and robots we discussed in class. Write down the modelview matrices and also the graph that includes all the components. A traversal of the trees should essentially render and animate the rickshaw.
   b. (10) Now let us say you are animating a street scene in Asia replete with these rickshaws. There are many plying the streets ferrying passengers all over the cities. Let the scene be a 2-way street chock-full of rickshaws plying in both directions. How will you represent this dynamic scene chock-full of rickshaws.
D. (70) Textures and Particle Systems
   a. (10) Consider a particle that is moving at constant speed as shown below bouncing elastically from surface to surface. Plot the trajectory of the particle.
   b. (10) What are the challenges of rendering particle systems? List them and explain how they impact the rendering performance and quality. What problems do the mechanisms of Texture Buffer Objects and TransformFeedbackVayings address? We covered these in class the last time we met. Therefore, I am looking for a very very short description (this part is only worth 3 points max).
   c. (10) Write pseudo-code to create a 256x256 pixel brick wall with each “brick” 2x2 pixel wide. This brick wall can have all the bricks in each row to be perfectly aligned.
   d. (10) Now give me psuedo-code which will give me a realistic brick wall. Each row of the wall is “shifted” and the rows of bricks are not aligned anymore.
   e. (10) Make the wall even more realistic by adding roughness. What techniques will you use? Provide pseudo-code.
   f. (10) How is bump mapping achieved? Sketch the fragment and vertex shaders that one would use for rendering bump mapped spheres. Qualitative description will suffice.
   g. (5) Explain the need for magnification and minification filters?
   h. (5) What are mip-map filters best for?
E. (30) All about spheres -

1. (25) Implement the following code segments. What does they really do? Explain the steps. Implement the shaders in the usual manner of having a front end WebGL program. Include the code and a picture of what is produced from the code.
   a. /* vertex shader */ void main()
      { gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
        gl_TexCoord[0] = gl_MultiTexCoord0;
      }
   b. /* fragment shader */ uniform sampler2D tex0;
      uniform float border; // 0.01
      uniform float circle_radius; // 0.5
      uniform vec4 circle_color; // vec4(1.0, 1.0, 1.0, 1.0)
      uniform vec2 circle_center; // vec2(0.5, 0.5)
      void main (void)
      {
        vec2 uv = gl_TexCoord[0].xy;
        vec4 bkg_color = texture2D(tex0,uv * vec2(1.0, -1.0));
        // Offset uv with the center of the circle.
        uv -= circle_center;
        float dist = sqrt(dot(uv, uv));
        if ( (dist > (circle_radius+border)) || (dist < (circle_radius-border)) )
          gl_FragColor = bkg_color;
        else
          gl_FragColor = circle_color;
      }

2. (5) Speculate where most time expended in the code? What happens when I change the radius and border to be very large? Or when change them to be very small?