Name ____________________________________

GPU Programming
EE 4702-1
Midterm Examination

Wednesday, 12 November 2014 9:30–10:20 CST

Problem 1	__________	(18 pts)
Problem 2	__________	(12 pts)
Problem 3	__________	(24 pts)
Problem 4	__________	(12 pts)
Problem 5	__________	(14 pts)
Problem 6	__________	(20 pts)
Exam Total	__________	(100 pts)

Alias ____________________________________

Good Luck!

Problem 1: [18 pts] Appearing below is a simplified version of the vertex shader `vs_main` from Homework 4. One way to reduce the amount of computation it must perform is by pre-computing values and making them available as `vs_main` inputs. For example, rather than compute `radial_idx` each time, it can instead be pre-computed and stored in the `w` component of `gl_Vertex` or it can be stored in a user-defined vertex shader input named `radial_idx`.

```
void vs_main() {
  int bidx = gl_Vertex.x;              int ti = gl_Vertex.y;

  int radial_idx = bidx * opt_segments + ti;
  float theta = (1.0/opt_segments) * radial_idx * omega;

  vec3 pos1 = balls_pos[bidx-1].xyz;    vec3 pos2 = balls_pos[bidx].xyz;
  vec3 v12 = pos2.xyz - pos1.xyz;
  vec3 p = pos1 + float(ti) / opt_segments * v12;

  vec3 va = spiral_radius * normalize( vec3(0,v12.z,-v12.y) );
  vec3 vb = spiral_radius * normalize( cross(v12,va) );

  vec3 radial = va * cos(theta) + vb * sin(theta);
  vec3 p_outer = p + radial;           vec3 p_inner = p + 0.5 * radial;
```

Consider two candidates for new vertex shader inputs: `va/vb` (count these as one candidate) and `theta`.

(*a*) Estimate the amount by which each candidate would reduce the amount of computation to be performed **by the vertex shader**.

☐ Computation reduction by making `va/vb` an input:

☐ Computation reduction by making `theta` an input:

(*b*) One of the candidates would result in a large increase in CPU to GPU data transfer. Which one is it, and why?

☐ Candidate causing a big increase in CPU to GPU data transfer:

☐ Reason for **big** increase:

(*c*) In the code above there is a much better candidate for a new vertex shader input other than `va/vb` or `theta`. It would reduce computation by a significant amount and would not have a big impact on data transfer. *Hint: it's not a variable, but an expression, or several expressions.*

☐ The much better candidate for an input attribute is:

Problem 2: [12 pts] Appearing below is the fragment shader from the solution to Homework 4. Recall that this shader does not render the spiral surface in places where the texture is dark, giving the appearance of holes.

(*a*) Modify the shader code so that the texture itself is only applied to the front side of the spiral, and so that the holes still appear on **both** sides. *Hint: Can be done with a line or two of code.*

☐ Modify so holes are on both sides, but texture just on front.

```
void fs_main() {
  vec4 color = is_edge ? gl_FrontMaterial.ambient :
        gl_FrontFacing ? gl_FrontMaterial.diffuse : gl_BackMaterial.diffuse;

  vec4 texel = is_edge ? vec4(1,1,1,1) : texture(tex_unit_0,gl_TexCoord[0].xy);

  bool hole = texel.r + texel.g + texel.b < 0.05;
  if ( hole ) discard;

  gl_FragColor = texel * generic_lighting(vertex_e, color, normalize(normal_e));

  gl_FragDepth = gl_FragCoord.z;
}
```

(*b*) The fragment shader code below, based on Homework 4, is supposed to use different colors for the front, back, and edge of the spiral. The code below has an error and the commented-out line shows the correct code. How will this error change the appearance of the spiral?

☐ With this error spiral will appear...

```
void fs_main() {
  vec4 our_color = is_edge ? gl_FrontMaterial.ambient :
            gl_FrontFacing ? gl_FrontMaterial.diffuse : gl_BackMaterial.diffuse;

  vec4 texel = is_edge ? vec4(1,1,1,1) : texture(tex_unit_0,gl_TexCoord[0].xy);
  bool hole = texel.r + texel.g + texel.b < 0.05;
  if ( hole ) discard;

  //  gl_FragColor = texel * generic_lighting(vertex_e, our_color, normalize(normal_e));
  gl_FragColor = texel * our_color;

  gl_FragDepth = gl_FragCoord.z;
}
```

Problem 3: [24 pts] Answer each question below.

(*a*) An object at point P_1 is moving at velocity v_1 just before collision with a wall with normal $\hat{w}$. The collision is completely elastic, meaning the object's speed (the magnitude of velocity) will not change. Write an expression for the velocity after the collision?

☐ Expression showing velocity after collision:

(*b*) An object at P_1 is connected to an object at P_2 by an ideal spring having a relaxed distance of d and a spring constant h. Write an expression for the force on P_1 due to this spring.

☐ Expression giving force on P_1 due to spring:

(*c*) Point P_1 in 3D space has homogeneous coordinate $(3, 4, 6, 0.5)$. What is the corresponding Cartesian (ordinary every day) coordinate?

☐ Cartesian coordinate is:

(*d*) Homogeneous coordinates make it possible to express certain transformations using matrix multiplication that would not be possible with ordinary Cartesian coordinates. Name one such transformation.

☐ Transformation that's **not** possible using matrix multiplication with Cartesian coordinates:

Problem 4: [12 pts] Answer each question below.

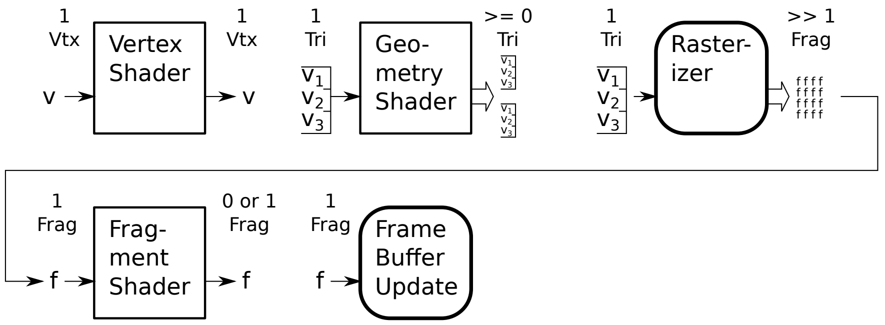
(*a*) Our user's eye is at location (x, y, z) in our coordinate system. How do we provide this information to OpenGL (using the compatibility profile)? Be specific.

☐ User's location given to OpenGL by ...

(*b*) The series of transformations performed by OpenGL (and shaders) maps 3D coordinates in our coordinate space to *window space* which are the 2D window coordinates corresponding to pixel locations. This transformation preserves coordinates' z component. Why is the z component needed, and how is it used?

☐ The z component is used for ...

Problem 5: [14 pts] The diagram below shows a simplified form of the OpenGL rendering pipeline.



(*a*) The diagram shows 16 f's at the output of the rasterizer stage, but that's not necessarily the actual number.

☐ What does the actual number of f's depend on?

☐ How could one increase the number of f's without changing the input, $(V_1V_2V_3)$.

(*b*) The boxes with rounded corners are part of what's called the *fixed functionality*. What does that mean, and why are those stages part of it.

☐ Fixed functionality means . . .

☐ The Rasterizer and Frame Buffer Update are fixed functionality because . . .

Problem 6: [20 pts] Answer each question below.

(*a*) The code below specifies vertices using a triangle strip. Modify the code so that it renders the same triangles using individual triangles.

☐ Modify code to use individual triangles ☐ without changing what's rendered.

```
glBegin(GL_TRIANGLE_STRIP);

for ( int i=0   ;    i<num_v   ;    i++    )
  {


    glVertex3fv( coord[i] );


  }
glEnd();
```

(*b*) There is an important difference between specifying a color using `glColor` and `glMaterial`. One of them allows each vertex to have its own color, the other allows different properties to be set, such as ambient, diffuse, and specular but all vertices in a rendering pass must share these colors. *Note: This question applies only to the compatibility profile.*

☐ The command that allows each vertex to have its own color is:

☐ Why can't each vertex have its own set of color properties using either of these commands?

(*c*) Describe how the `GL_LINEAR_MIPMAP_LINEAR` minification method computes a filtered texel. Indicate how many texels are read from texture images, which images are read, and how they are combined.

☐ Number of texture images read:

☐ Number of texels read:

☐ How they are combined: