Name ______________________________

GPU Programming
EE 4702-1
Final Examination

Tuesday, 5 December 2017   12:30–14:30 CST

Problem 1	______	(15 pts)
Problem 2	______	(20 pts)
Problem 3	______	(30 pts)
Problem 4	______	(35 pts)
Exam Total	______	(100 pts)

Alias ______________________________

Good Luck!

Problem 1: [15 pts] Appearing below is code based on Homework 5 (the tile cloud). First is code that launches the rendering passes, followed by the shaders. The shaders below were written to work with the rendering pass launched by `case 1`. In `case 3`, which is incomplete, the rendering pass input primitives are lines instead of triangles. Complete the `case 3` code and modify the shaders to work with your completed `case 3` code.

☐ Complete the `case 3` code. Use abbreviations such as `glV` (for `glVertex4f`), `glC`, and `glN`.

```
case 1: {  pShader_Use use(s_hw05_tiles_1);
           glBegin(GL_TRIANGLES);
           for ( Tile* tile: tiles ) {  glColor4fv(tile->color);    glVertex4fv(tile->pt_00);
                                        glColor4fv(tile->tact_pos); glVertex4fv(tile->ax);
                                                                    glVertex4fv(tile->ay); }
           glEnd();  } break;
case 3: {  pShader_Use use(s_hw05_tiles_3);
           glBegin(GL_LINES);   //  <--- DON'T FORGET, LINES
           for ( Tile* tile: tiles ) {



           }
           glEnd();  } break;
```

☐ Modify shader code (below) to work with `case 3` (above). Look at ☐ type of primitive (above), ☐ interface blocks, ☐ shader routines, and ☐ layout declarations.

```
out Data_to_GS { vec4 vertex_o;    vec4 color;       };

void vs_main_tiles_3() {  // Vertex shader routine.                // CHANGE/ADD SOMETHING
  vertex_o = gl_Vertex;
  color = gl_Color;
}

in Data_to_GS { vec4 vertex_o;      vec4 color;        } In[3];     // CHANGE/ADD SOMETHING

layout ( triangles ) in;                                            // CHANGE/ADD ONE OF
layout ( triangle_strip, max_vertices = 4 ) out;                    // THE LAYOUT DECLARATIONS

void gs_main_tiles_3() {   // Geometry shader routine.              // CHANGE/ADD SEVERAL THINGS

  vec4 pt_00 = In[0].vertex_o;

  vec4 ax_o = vec4(In[1].vertex_o.xyz,0);

  vec4 ay_o = vec4(In[2].vertex_o.xyz,0);

  vec4 tact_pos = In[1].color;

  color = In[0].color;

  vec4 vtx_o[4];
  vtx_o[0] = pt_00;          vtx_o[1] = pt_00 + ax_o;
  vtx_o[2] = pt_00 + ay_o;   vtx_o[3] = pt_00 + ay_o + ax_o;
  // The code below does not need to be changed and so isn't shown.
```

Problem 2: [20 pts] Appearing below is the `render_tiles` routine from Homework 5.

(*a*) Estimate the amount of data sent from the CPU to the GPU for the rendering pass started by the code below. Use the following symbol: n, the number of tiles.

```
  case 1: {
      pShader_Use use(s_hw05_tiles_1);
      glUniform2i(1, opt_tryout1, opt_tryout2);
      glUniform1i(2, light_state_get());
      glUniform1f(3, world_time);

      glBegin(GL_TRIANGLES);
      for ( Tile* tile: tiles ) {
          glColor4fv(tile->color);
          glVertex4fv(tile->pt_00);
          glColor4fv(tile->tact_pos);
          glVertex4fv(tile->ax);
          glVertex4fv(tile->ay);
        }
      glEnd();
    }
    break;
```

☐ Amount of data, in bytes:

(*b*) The vertex shader below is used with the `case 1` code above. In terms of n, how much data is read by this vertex shader for a rendering pass?

```
void vs_main_tiles_1() {
  vertex_o = gl_Vertex;
  color = gl_Color;
}
```

☐ Total data read by shader above for a rendering pass:

☐ Explain why the amount of data read by the vertex shader might be different than the amount of data sent from CPU to GPU when executing the `case 1` code.

Problem 2, continued: The rendering pass below provides the same data to the shaders as the one in the previous part, but there are significant differences.

```
#define TO_BO(name,num,update)                                               \
  glBindBuffer(GL_ARRAY_BUFFER,bos_tiles[num]);                              \
  if ( update ) glBufferData                                                 \
    (GL_ARRAY_BUFFER, name.size()*sizeof(name[0]), name.data(), GL_STREAM_DRAW); \
  glBindBufferBase(GL_SHADER_STORAGE_BUFFER,num,bos_tiles[num]);

      pShader_Use use(s_hw05_tiles_2);
      glUniform2i(1, opt_tryout1, opt_tryout2);
      glUniform1i(2, light_state_get());
      glUniform1f(3, world_time);

      TO_BO(pt_00,     1, pt_00_data_stale);
      TO_BO(ax,        2, axes_data_stale);
      TO_BO(ay,        3, axes_data_stale);
      TO_BO(color,     4, color_data_stale);
      TO_BO(tact_pos,  5, tact_data_stale);
      pt_00_data_stale = axes_data_stale = color_data_stale = tact_data_stale = false;

      glDrawArrays(GL_POINTS,0,tiles.size());
```

(*c*) Explain why the code above would be more efficient when rendering the very first frame and why it might be more efficient later in the execution, depending on how the tiles are used.

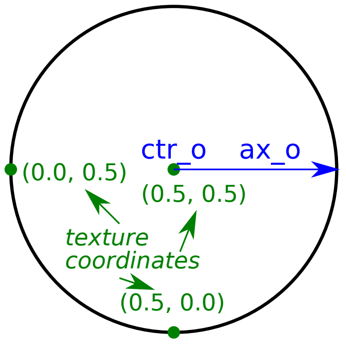
☐ More efficient for the very first frame because:

☐ More efficient later in execution because . . .

☐ This efficiency can be assumed by use of variable such as . . . because . . .

Problem 3: [30 pts] The code in this problem is similar to the Homework 5 tile code except that it will be used to render discs.

(*a*) Complete the geometry shader below so that it emits primitives for a disc (a filled-in circle) writing the geometry shader outputs shown. The provided shader code reads the shader disc center, `ctr_o`, disc normal, `nz_o`, and a vector to a point on the circumference, `ax_o` (see diagram), all in object space. The shader output includes `tcoord`, a texture coordinate. Assign this consistent with the texture coordinates shown on the diagram. Assume that the shader output primitive, a triangle fan, will work correctly even though it is not included in OpenGL Shading Language 4.5. A triangle fan makes the solution simpler. *Note: This triangle fan discussion was intentionally omitted from the original exam.*



☐ Emit primitives for the disc, make sure it's filled in.

☐ Don't forget to: ☐ Set `max_vertices`, ☐ set `normal_e`, ☐ set `tcoord`, and of course ☐ set `gl_Position`.

```
out Data_to_FS { flat vec3 normal_e;  vec3 vertex_e;  vec2 tcoord;};

layout ( points ) in;
layout ( triangle_fan, max_vertices =       ) out;  // <--- FILL IN

void gs_main_disc() {
  int vertex_id = In[0].vertex_id;
  vec4 ctr_o = ctrs[vertex_id];
  vec3 ax_o = axs[vertex_id].xyz;
  vec3 nz_o = nzs[vertex_id].xyz;

  const int slices = 10;
  const float pi = 3.1415926536;
  const float delta_theta = 2 * pi / slices;
// Abbrevs: glmvp, gl_ModelViewProjectionMatrix; glmv, gl_ModelViewMatrix; gln, gl_NormalMatrix




  for ( int i=0; i<slices; i++ )
    {
      float theta = i * delta_theta;
      float costh = cos(theta);
      float sinth = sin(theta);






    }
}
```

Problem 3, continued:

(*b*) Appearing below is the fragment shader for the code above. If variable `nevermind` were `true` the fragment shader would not write a fragment, but it's set to `false` in the code. (The `discard` keyword returns from the fragment shader without writing a fragment.)

The primitives emitted by the geometry shader (if solved correctly) will render a 10-sided polygon, which is not exactly a disc (circle). Modify the code so that it emits a perfect disc based on the largest circle that can fit inside the polygon. (For partial credit, a circle of radius 0.4 in texture coordinate units.) Use texture coordinates to determine whether a fragment is in the circle.

☐ Assign `radius` the correct value in terms of `slices`.

☐ Set `nevermind` so that a fragment is discarded if it's outside a radius-`radius` circle based on `tcoord`.

```
void fs_main_disc() {
  const int slices = 10;
  const float pi = 3.1415926535;
  const float delta_theta = 2 * pi / slices;




  float radius = 0.4;                        // CHANGE TO CORRECT VALUE.




  const bool nevermind = false;             // CHANGE FOR PERFECT CIRCLE




  if ( nevermind ) discard; // Don't write fragment, exit the shader.

  vec4 texel = texture(tex_unit_0,tcoord);   // NO NEED TO CHANGE THIS CODE.
  vec4 color = colors[vertex_id];
  gl_FragColor = texel * generic_lighting(vertex_e, color, normal_e);
  gl_FragDepth = gl_FragCoord.z;
}
```

(*c*) The inputs to fragment shader `fs_main_disc` appear below. Explain the implications of moving the `flat` qualifier as described below.

```
in Data_to_FS  {  flat vec3 normal_e;   flat int vertex_id;
                  vec3 vertex_e;        vec2 tcoord;              };
```

☐ Explain impact on ☐ correctness and ☐ efficiency if `flat` were removed from `normal_e` and `vertex_id`.

☐ Explain impact on ☐ correctness and ☐ efficiency if `flat` were added to `vertex_e` and `tcoord`.

Problem 4: [35 pts] Answer each question below.

(*a*) Explain how the depth (z-buffer) test is used. Provide an example in which sorting primitives by eye distance makes the depth test unnecessary, and another example in which even with sorting a depth test is necessary for proper rendering.

☐ Explain how depth test used.

☐ Example where sorting makes depth test unnecessary. *Hint: Example can have two primitives.*

☐ Example where depth test necessary even with sorting. *Hint: Example can have three primitives.*

(*b*) Appearing below are sample uses of two procedures related to the stencil buffer. Explain what each one does in general (not necessarily in the example).

```
glStencilFunc(GL_EQUAL,4,-1);
glStencilOp(GL_REPLACE,GL_KEEP,GL_KEEP);
```

☐ The `glStencilFunc` procedure is used to ….

☐ The `glStencilOp` procedure is used to ….

(*c*) In general, why doesn't it make sense to access a texture in a vertex shader?

☐ Bad idea to access a texture in a vertex shader because . . .

(*d*) How is eye space defined? Describe the transformations that need to be applied to map from object space to eye space. Illustrate your answer using a sample scene.

☐ Defining features of eye space are . . ..

☐ In the following scene to transform from object to eye space the modelview matrix is used, which . . .

(*e*) An NVIDIA GPU has 10 SMs. Consider a kernel which evenly divides work among its threads, the usual assumption made in class. Further assume that there is a large amount of work. Let $t(G)$ denote the execution time when the kernel is launched with G blocks. In all cases the block size is 1024 threads.

Let $a = t(10)$, the time when launched with 10 blocks on the 10-SM GPU.

☐ Find an expression for $t(5)$ in terms of a.

☐ Find an expression for $t(15)$ in terms of a.

☐ Find an expression for $t(20)$ in terms of a.

(*f*) Draw a sketch showing the shadow volume corresponding to a triangle. Include the triangle, the light source, some object in the shadow and some object seen through the shadow.

☐ Show: ☐ the light, ☐ triangle, ☐ shadow volume for the triangle, ☐ shaded object, ☐ object seen through shade.