# EECS 700: Exam # 2

Thursday, December 3, 2014

Print Name and Signature________________________________________

The rules for this exam are as follows:

- **Write your name on the front page of the exam booklet. Initial each of the remaining pages in the upper-right hand corner. Sign the front of the exam booklet. Your exam will not be graded if you have not signed the front page of the booklet.**
- The exam has 4 questions and is 6 pages long (including this page). Be sure you have all of the pages before beginning the exam.
- This exam will last for 75 minutes.
- Show **ALL** work for partial/full credit. This includes any definitions, mathematics, figures, etc.
- The exam is closed book and closed notes.
- You are allowed to use a calculator on the exam provided you use it to perform arithmetic computations only and show all of your work on the exam.
- No collaboration of any kind is allowed on the exam.

**Scores:**

| | |
|---|---|
| 1. _______ (20 pts) | 3. _______ (20 pts) |
| 2. _______ (20 pts) | 4. _______ (20 pts) |

**Total:** _______ (80 points)

**Grade:** _______

1. (20 points)

   (a) (15 points) Consider solving the following 1D Poisson problem:

$$-\frac{d^2u}{dx^2} = 1, \forall x \in (0,1), \text{with } u(0) = u(1) = 0.$$

   Let $x_j = j/n$ for $j = 0, 1, \ldots, n$ denote a set of $n+1$ uniformly-spaced mesh points on $[0,1]$. Derive the linear system that results from using the finite element method to discretize the boundary value problem. Use the standard piecewise linear "hat" basis functions for your derivation. **You do NOT need to solve the linear system.**

   (b) (5 points) Which linear solver should be used to efficiently solve the linear system derived above?

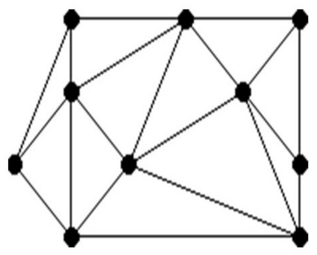
2. (20 points)

   (a) (15 points) Consider performing parallel matrix-vector multiplication of an $n \times n$ matrix $A$ with an $n \times 1$ vector $x$. **Give a pseudocode for parallel matrix-vector multiplication on $n$ processes using rowwise 1-D partitioning.** Consider the case in which $A$ is partitioned among $n$ processes so that each process stores one complete row of the matrix. Similarly, $x$ is distributed such that each process owns one of its elements. **Be sure to specify which MPI commands should be used for each step in your pseudocode. It's sufficient to give only the name of the command at each step.**

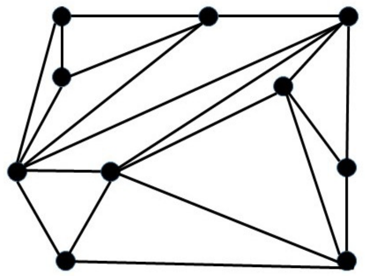
   (b) (5 points) What is the major difference in the basic communication operation required for performing parallel matrix-vector multiplication on $n$ processes when columnwise 1-D partitioning is used?

3. (20 points)

(a) (5 points) Label each of the two meshes below as a Delaunay mesh or a non-Delaunay mesh.



(a)



(b)

Figure 1: Two triangulations of a set of points

(b) (5 points) In the case of the non-Delaunay mesh in (a), explain why it is a non-Delaunay triangulation.

(c) (5 points) Specify the major goal of mesh generation for (i) engineering versus (ii) computer graphics applications.

(d) (5 points) What geometric primitive should be used to draw triangular meshes in an efficient manner with OpenGL? Justify your choice.

4. (20 points) **NOTE: This question takes two pages.**

   (a) (10 points) What does the following snippet of OpenGL code do? Label each line of the code. Assume proper initialization elsewhere.

```
static const GLfloat vertices[] = {
  -1.0f, -1.0f,  1.0f,
   1.0f, -1.0f,  1.0f,
   1.0f,  1.0f,  1.0f,
  -1.0f,  1.0f,  1.0f,
  -1.0f, -1.0f, -1.0f,
   1.0f, -1.0f, -1.0f,
   1.0f,  1.0f, -1.0f,
  -1.0f,  1.0f, -1.0f,
};

static const GLfloat colors[] = {
  1.0f, 0.0f, 0.0f,
  0.0f, 1.0f, 0.0f,
  0.0f, 0.0f, 1.0f,
  1.0f, 1.0f, 1.0f,
  1.0f, 0.0f, 0.0f,
  0.0f, 1.0f, 0.0f,
  0.0f, 0.0f, 1.0f,
  1.0f, 1.0f, 1.0f,
};

static const GLushort indices[] = {
  0, 1, 2,
  2, 3, 0,
  3, 2, 6,
  6, 7, 3,
  7, 6, 5,
  5, 4, 7,
  4, 5, 1,
  1, 0, 4,
  4, 0, 3,
  3, 7, 4,
  1, 5, 6,
  6, 2, 1,
};

glGenBuffers(1, ebo);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ebo[0]);
```

```
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices,
    GL_STATIC_DRAW);

glGenVertexArrays(1, vao);
glBindVertexArrays(vao[0]);

glGenBuffers(1, vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices) + sizeof(colors), NULL,
    GL_STATIC_DRAW);
glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(vertices), vertices);
glBufferSubData(GL_ARRAY_BUFFER, sizeof(vertices), sizeof(colors),
    colors);

glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, NULL);
glEnableVertexAttribArray(0);

glBindVertexArray(vao[0]);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ebo[0]);

glDrawElements(GL_TRIANGLES, 12, GL_UNSIGNED_SHORT, NULL);
```

(b) (6 points) Specify two ways in which the above OpenGL code could be improved. **Do NOT rewrite the code.**

(c) (4 points) **TRUE or FALSE?** Vertex and fragment shaders are run on the GPU.