Full Name: ______________________________

Andrew ID: ______________________________

15-462/662, Fall 2017

Midterm Exam

October 18, 2017

Instructions:

- This exam is **closed book, closed notes, closed neighbor, closed cell phone, closed telepathy, closed internet**.
- You may however use a single 3in x 3in sticky note (or piece of paper) with any information you like written on both sides—-*except* for solutions to previous exams.
- If your work gets messy, please clearly indicate your final answer (by writing it in a box if possible).
- Partial credit will be awarded, *but only if we can understand your work!* So please try to write clearly, especially if you are uncertain about the final answer.

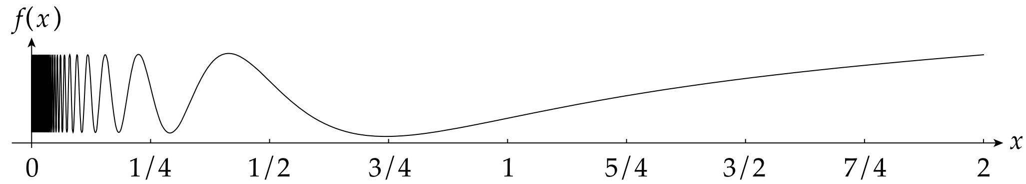
Problem	Your Score	Possible Points
1		25
2		25
3		25
4		25
EC		20
Total		100

1 (25 points) Getting the Party Started: Miscellaneous Short Problems

A. (5 points) Consider a function $f : \mathbb{R}^m \to \mathbb{R}^n$. What does it mean for f to be a *linear function*? How can you represent the *affine* function $f(x, y) = ax + by + c$ as a linear function in homogeneous coordinates?

B. (5 points) In your own words, what is *aliasing?* Give two examples of aliasing that show up either in graphics algorithms or in real life.

C. (5 points) **Sampling the Topologist's Sine Curve**



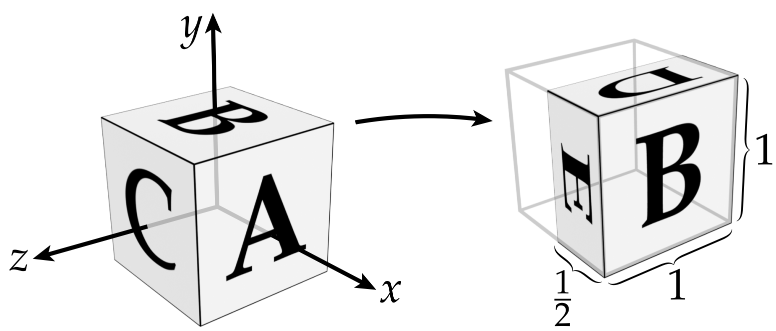
Consider the curve above, which is given by the function

$$f(x) := 2x\cos(\pi/x) + \pi\sin(\pi/x) + 4.$$

For each quarter-interval, *roughly* how many samples would you guess are needed to get an accurate reconstruction of this function (*e.g.*, using linear interpolation)? Why?

Suppose now that rather than linear interpolation, you want to generate a high-quality 1D texture representing this function (*i.e.*, a single strip of texels, with one texel per quarter interval). Other than sampling, how could you obtain intensity values for these texels? **Bonus points:** give the exact value for the first texel in this image (as a fraction).

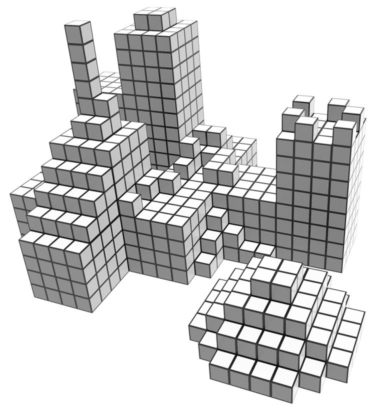
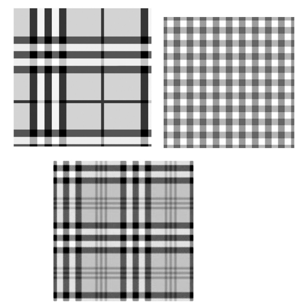
D. (5 points) Describe in words any sequence of transformations whose composition produces the overall transformation depicted in the figure below. The initial shape is an axis-aligned cube centered at the origin with all side lengths equal to 1; the gray outline on the right indicates the location of the original cube relative to the new cuboid.



E. (5 points) Suppose we have a surface with no boundary (*i.e.*, no holes) described by an explicit triangle mesh, and need to do many inside/outside tests. What is a simple implicit representation that makes these queries more efficient, and how might we perform the conversion from explicit to implicit? There are several reasonable solutions here; your solution should attempt to make the *queries themselves* as efficient as possible, *i.e.*, smallest amortized cost to do a single query. You do not need to write any code; just give a high-level outline.

F. (5 pts — EXTRA CREDIT) Your solution to the previous question may not be exact for query points very close to the original surface, *i.e.*, it may classify points that are "inside" as "outside" and vice versa. How can you modify your solution to provide exact queries near the surface? How can you make this fallback efficient?

2 (25 points) TartanCraft

In the massively multiplayer online game of *TartanCraft*™, players create 3D worlds out of cubes textured with lovely plaid patterns. Each cube is determined by a center location (i, j, k), as well as a index specifying which pattern to use.

A. (5 points) Consider the difference between rendering this scene using a rasterizer and a ray tracer: specifically think about the cost of determining which cube is visible at each screen sample in a rasterizer, or along each camera ray in a ray tracer. Do you think determining occlusion using the Z-buffer algorithm or via ray tracing is a better solution in this scenario? Why? You can assume that in the case of a ray tracer, you have a prebuilt spatial acceleration hierarchy. Also assume that the geometry is really a large collection of cubes (including those “deep inside” objects), rather than just those on the outer boundary.

B. (5 points) His Imperial Majesty Andrew Carnegie, Emperor of the Tartan Realm, wants a panoramic view of everything going on in the world of *TartanCraft* from a *fixed* viewpoint far above the ground. Suppose that cubes from all players are being streamed over the network to his machine, and he wishes to watch the world as it is being built. Cube data can arrive at any time, in any order. You can assume that cubes are created, but never destroyed. What rendering strategy might Emperor Carnegie use to render these millions of cubes: rasterization or ray tracing? Why? (Think again about how visibility will be determined.)

C. (5 points) Suppose that in one of the scenarios above we chose to use ray tracing rather than rasterization. Given that we're always rendering cubes, what's a small optimization we can make that (slightly) speeds up our ray intersection tests?

D. (5 points) Again suppose we're using ray tracing, rather than rasterization. What kind (or kinds) of spatial acceleration data structure(s) would you use for our little world made of cubes, and why?

E. (5 points) Finally, we have to gift-wrap our cubes with lovely plaid patterns (of the kind shown above). Given that we're rendering exclusively in plaid, what kind of texture magnification filter would you use to nicely preserve the appearance of these textures?

3 (25 points) Validating Meshes

An important part of writing robust code is verifying the validity of the input. Suppose you are given an adjacency list representation of a triangle mesh and are asked to convert it to a halfedge mesh. In particular, you are provided with the data:

```
double pts[nVertices][3]; // coordinates (x,y,z) for each vertex
int tris[nTriangles][3]; // triangles as triples of 0-based indices into pts
```

A. (6 points) What must be true about the `pts` list in order to convert this data into a valid halfedge mesh?

B. (6 points) Outline a simple strategy for simultaneously verifying that (i) no edge in the given mesh is nonmanifold, and (ii) all triangles have a consistent orientation (or "winding"). What is the cost of your strategy, assuming you have an efficient data structure that can map a *ordered* pair of integers to a stored integer value in $O(1)$ time (*i.e.*, a hash table / associative array)? You do not have to write any code.

C. (7 points) Suppose now that your code is running on the deep space probe *New Horizons II* which has been taking 3D scans of objects floating around in the Kuiper belt. You already have a nicely scanned and reconstructed halfedge mesh, stored in the data structure below. However, in outer space it is quite common for high-energy *cosmic rays* to hit the RAM of your machine, randomly invalidating bits. Write a routine that does an exhaustive consistency check on your halfedge mesh—a mesh that passes this test should describe a valid polygonal (though not necessarily triangular) surface. You may assume that a preliminary check has already been run to ensure that all pointers point to some valid element of the mesh, *i.e.*, not NULL or some totally random address. (Note that you do *not* need to check for non-manifoldness, since a halfedge mesh cannot represent a nonmanifold mesh!)

```
struct HalfedgeMesh
{
   vector<Halfedge> halfedges;
   vector<Vertex>   vertices;
   vector<Edge>     edges;
   vector<Face>     faces;
};
```

```
struct Halfedge
{
   Halfedge* t; // twin
   Halfedge* n; // next
   Vertex*   v; // vertex
   Edge*     e; // edge
   Face*     f; // face
};
```

```
struct Vertex
{
   Halfedge* h;
   Vector3D coords;
};
```

```
struct Edge
{
   Halfedge* h;
}
```

```
struct Face
{
   Halfedge* h;
}
```

```
bool isValid( HalfedgeMesh* mesh )
{
```

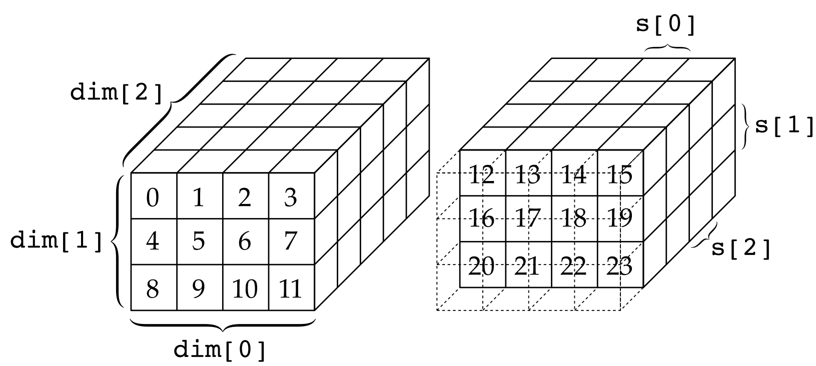
```
}
```

D. (6 points) Sending data across outer space is expensive. Below you are given the bare minimum encoding for the connectivity of two different meshes: one in adjacency list format, the other as the list of "next" pointers for the halfedges in a halfedge mesh. The twin pointers are encoded by a simple rule: the twin of a halfedge with *even* index i is $i+1$; the twin of a halfedge with *odd* index j is $j-1$ (so that consecutive even and odd indices are paired). Draw a picture of each mesh, and indicate whether each one is manifold (and why, in the case of a nonmanifold mesh). How many polygons does each mesh encode? How much storage is needed for each mesh?

```
                    NEXT
0 1 2         0   ---> 2
3 1 4         1   ---> 11
1 3 5         2   ---> 4
6 7 8         3   ---> 7
              4   ---> 0
              5   ---> 9
              6   ---> 1
              7   ---> 8
              8   ---> 3
              9   ---> 10
              10  ---> 5
              11  ---> 6
```

4 (25 points) Generalized Sampling

Uh oh. Your beloved pet terrier (named "Scotty", oddly enough) ate your most recent quiz for 15-462. To recover the solution, you take Scotty to get a CT scan, which lets you peek inside your trusted companion.



A. (5 points) Your first task is to implement a routine that simply looks up the sample value stored at a given integer index. In particular, the data is given as a linear array `data` of scalar (*i.e.*, grayscale) values. The main challenge here is translating a requested 3D index (i, j, k) into a single index I corresponding to the location of the requested sample value. The dimensions of the data are specified by the array `dim[3]`, which gives the number of samples along the x, y, and z axes, respectively. The data itself is packed into the linear array in *x-minor order*. In other words, incrementing by 1 moves along the x-axis; incrementing by `dim[0]` moves along the y-axis, and so forth (see the figure above for an example). For indices outside the array dimensions, you should just return zero. All indices start at zero (not one).

```
double lookup3D( double* data, // sample values
                 int dim[3],   // number of samples along each axis
                 int i, int j, int k ) // requested sample index
{









}
```

B. (6 pts) Since the CT scan is fairly low-resolution, you will need to upsample it using trilinear sampling. However, due to the nature of the hardware that measured this signal, the data is sampled at equal-size intervals in x and y, but at much coarser intervals in z. In other words, the distance between samples is **NOT** the same along all three axes—these distances are provided in the array `s[3]`. Your routine should take a sample location (x, y, z) and return the interpolated value at this point. You must make use of the routine `lookup3D` from the previous part (whether or not you completed this question). Since this routine returns zero for values outside the domain, you do not need to worry too much about special handling of values near the boundary. The center of grid cell $(0, 0, 0)$ is at the origin.

```
double sampleTrilinear( double* data, // sample values
                        int dim[3],   // number of samples along each axis
                        int s[3],     // spacing between samples along each axis
                        double p[3] ) // coordinates of the sample point
{
```

```
}
```

C. (4 pts) Based on the current treatment of values near the boundary, what will the edges of the interpolated signal look like? How might the interpolation strategy be modified to avoid this behavior?

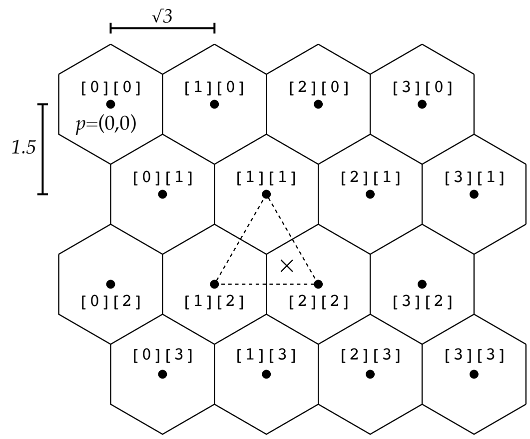
D. (4 pts) Suppose that instead of a static 3D scan you now have a movie showing the CT scan over time, stored as a 4D grid (3D + time). How many samples would you need to evaluate *quadrilinear* sampling of this data, and what would your basic strategy be? (You do not need to write any code here.)

E. (6 pts) Implement a method which, given a query point p, interpolates the three values u, v, w at the corners a, b, c (respectively) of a triangle in 2D using barycentric interpolation. If the query point is outside the triangle, you should return zero. All coordinates are specified via a `Vector2D`, which has coordinates `p.x` and `p.y`, as well as all the usual vector operations (addition, subtraction, dot product, cross product, *etc.*). You may use whatever names you like for these methods, as long as the meaning is clear.

```
double sampleBarycentric( Vector2D p, // sample point
                          Vector2D a, Vector2D b, Vector2D c, // triangle corners
                          double u, double v, double w ) // values at corners
{

}
```

EXTRA CREDIT: *This final question is challenging and should be completed for* extra credit only *(you are not required to answer it in order to get full points. You may wish to try it after completing the others.*



(15 pts) Once the homework was located inside your dog, the doctor took a high-resolution image of your quiz solution using a special, high-res 2D camera. To reduce directional aliasing, this 2D sensor has pixels arranged in a regular *hexagonal* grid rather than a rectangular grid. The grid is comprised of equilateral triangles with unit side length, which means the horizontal spacing between sample centers is $\sqrt{3}$ and the vertical spacing is 1.5; the center of grid cell $(0,0)$ is at the origin. To reconstruct the image, you must now interpolate values on this grid. To do so, you will perform barycentric interpolation using the three sample values closest to a given point (marked by an "X" in the image above). This time, you may assume that a method `lookup2DHex( data, i, j )` has already been defined, which returns the value of `data` at index (i,j)—cells are indexed as indicated in the example above. You do not need to worry about the width or height of the grid; values outside the valid range will just return zero. You may use the method `sampleBarycentric` defined in the previous question, though you may find it easier to just compute the interpolation directly.

```
double sampleHexGrid( double** data, // sample values
                      int nCols,     // number of horizontal samples
                      int nRows,     // number of vertical samples
                      Vector2D p )   // sample point
{
```

```
}
```