The Honor Code is the University's statement on academic integrity written by students in 1921. It articulates University expectations of students and faculty in establishing and maintaining the highest standards in academic work:

1. The Honor Code is an undertaking of the students, individually and collectively:
   a. that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;
   b. that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

2. The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.

3. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

I acknowledge and accept the Honor Code.

NAME (Please Print):

Signature:

Note: This is exam is open-book, open-notes, open-laptop, but closed-network.

The exam consists of 5 questions. Each question is worth 20 points. Please answer all the questions in the space provided, overflowing on to the back of the page if necessary.

This exam has been designed to take 2 hrs. However, you have 3 hours to complete the exam.
1. [20 points] Graphics in General

1A [5 points, 3 parts]. What is your favorite font? Look at the font used to print this exam. Does the font on this page have serifs? Look at the font used in Pat’s slides. Does that font have serifs?

Any valid font is an acceptable answer as their favorite font. The exam's font has serifs. The font used in Pat’s slides does not have serifs.

- 2 points for an incorrect answer to one part.

1B [5 points, 2 parts]. In graphics we use both point and area light sources. What visual effect is caused by an area light source? Graphics systems like OpenGL only use direct lighting. More advanced systems use indirect lighting. What is the difference between direct and indirect lighting?

(+ 2 points)
Unlike an point light source, an area light source occupies a finite area in space. A point light source creates hard-edge shadows while an area light source casts soft shadows.

(+ 3 points)
Direct lighting simulates light that comes directly from light sources. Indirect lighting simulates light bouncing off from objects in the environment (i.e. neighbor objects). For example, it can be light reflected from nearby objects or light transmitted through nearby semi-transparent objects.
1C [5 points]. We use many different basis functions in computer graphics. Name five different basis functions that we discussed in this course.

+1 point for each correct basis function.

Commonly accepted answers:
- Haar wavelet basis
- Hermite basis functions
- Triangle basis
- Square basis
- Discrete cosine basis
- Bernstein polynomials Bézier basis functions

1D [5 points]. When we send a triangle to OpenGL, we specify information at the vertices. For example, position, normals, colors, texture coordinates, etc. When the GPU draws the triangle, the rasterization hardware interpolates the vertex information so that every fragment has a value. The hardware uses barycentric interpolation.

Here is the triangle. The value 6 is at (0,0), the value 3 at (1,0), and the value 3 at (0,1). Compute the interpolated value at the point (1/3, 1/3) using barycentric interpolation.

\[
A_{total} = 1 \times \frac{1}{2} = \frac{1}{2} \\
A_1 = \frac{1}{3} \times 1 \times \frac{1}{2} = \frac{1}{6} \\
A_2 = \frac{1}{3} \times 1 \times \frac{1}{2} = \frac{1}{6} \\
A_3 = \frac{1}{2} - \frac{1}{6} - \frac{1}{6} = \frac{1}{6} \\
\alpha_1 = \frac{1}{3} \\
\alpha_2 = \frac{1}{3} \\
\alpha_3 = \frac{1}{3} \\
\text{Answer} = \frac{1}{3} \times 3 + \frac{1}{3} \times 3 + \frac{1}{3} \times 6 = 4
\]

+ 3 points for demonstrating understanding of Barycentric interpolation
+ 2 points for the correct answer
2. [20 points] Color and Displays

Most color displays use 3 colors. Recently, 4 color displays are beginning to appear in televisions and cell phones.

2A [4 Points]. Why are most displays built using only three primary colors?

They need to mention something about the human visual system having three cones, about the color matching experiment, trichromatic vision, etc.

*If they just say that 3 primaries lets us see most of the perceivable colors without saying why, -1.*

-1 *if they say that it can display ALL perceivable colors, without giving the correct answer to part 2B.*

2B [4 Points]. What would be the advantage of building a display using 4 colors?

They should say that adding a 4th primary can enlarge the display gamut.

-1 *if they somehow suggest that a 3 color display could display ALL perceivable colors, but it's hard to get the correct primaries to do that, or something to this effect.*

-2 *if they say that it just adds "a dimension of richness" or "more vibrant colors" without explaining why this is or what this means.*

-2 *if the only reason they give is to make displays for tetrachromats.*

-3 *if they say that only tetrachromats would benefit from four primary displays.*

-1 *if they only mention HDR as a benefit or increased contrast with RGBW displays.*
2C [12 Points]. Most content such as images and videos contain three component RGB colors. Suppose we want to display an RGB image on a 4-color display. To be specific, say the display colors are Red, Green, Blue and Yellow. The RGB values for the Yellow are (YR, YG, YB). Given the color (R, G, B), give an algorithm to determine the display colors (Rd, Gd, Bd, Yd). Justify your algorithm.

I gave full credit for \((Rd, Gd, Bd, Yd) = (R - Yd*YR, G - Yd*YG, B - Yd*YB, Yd)\) if they have a method for choosing \(Yd\) such that none of the colors is negative.

If they don't say how to set \(Yd\), or the technique they give allows negative numbers, -2.

If they give the trivial answer with \(Yd = 0\), -6 points.

I usually gave partial credit even if the answer was wrong. I got a lot of answers, so depending on how far off the answer was, or how much explanation/work they showed, I took off -5, -6, -8 or -12.
3. [20 points]. Geometric Modeling

A very common format for geometric models is the points/polygons format. The well-known OBJ file format, which is read in by the STShape class, is an example of points/polygons format. Points/polygon format consists of an array of points. Each point is given by its index. The array of points contains the coordinates of each point. Each polygon is also given by an index. This index points to a linked list data structure that contains all the points that make up the polygon.

Points/polygon format is fine if we just want to draw the polygons. However, when modeling we often have to do other operations such as compute vertex normals, subdivide a mesh into a more detailed mesh, etc.

In class, we described a simple data structure for representing triangle meshes. Each triangle contains pointers to 3 vertices and 3 triangles.

```
typedef struct {
    Face *t;
    Point pt;
} Vert;

typedef struct {
    Face *t[3];
    Vert *v[3];
} Face;
```

3A [14 Points]. 3A [14 Points]. Give an algorithm for converting points/polygon format to a triangle mesh. Your algorithm must run in linear time. You can not assume that polygons are triangles in the original mesh, however, you MAY assume that polygons are convex. You will need to (i) come up with a way to convert the original polygons to triangles and (ii) describe how to convert the newly subdivided polygons to the format described above (HINT: think about using a data structure to associate edges and triangles they connect to).

+ 6: valid strategy for dividing polygons into triangles.
Common answers that were accepted:
- triangle fan
- define midpoint and draw edge to each vertex
  + 2 to 4 partial credit given for strategies that were close

+ 8 total for correct algorithm to convert mesh:
+ 2 partial credit given for incorrect/incomplete algorithms that did any of the following correctly:
  - added vertices to each face
  - added adjacent faces to each face
  - created vertices correctly
  + 2 to 4 partial credit given for using correct mapping structure and/or explaining its use correctly
3B [6 Points]. There exist points/polygon files that cannot be converted to triangle mesh data structure given above. Given 3 examples of topological situations that would prevent the points/polygon file to be successfully converted.

+6 points total, +2 for each correct answer.

commonly accepted answers:

- disjoint faces
- collinear vertices
- concave/non-convex (accepted even though given in part a)
- more than 2 faces connecting to an edge
- holes
- self intersecting polygons
- disjoint vertices

*Note: if gave multiple answers that were too similar (like multiple different concave cases or self intersecting cases) either no credit or partial credit (1 point) given for the repeated answers.

*Note: in some cases if their implementation was correct in part A but not what we expected, answers to part B may be correct in these cases.
4. [20 points] Signal Processing and Sampling

One of the biggest problems in computer graphics is aliasing. Aliasing causes many objectionable artifacts in computer-generated images.

4A [5 Points]. Suppose we sample a signal at some frequency $f_s$. $f_s$ is the number of samples per interval of time $T$. Suppose we have a signal in time composed of different frequencies. Some signals will cause aliasing, and some will not. What is the precise property of the signal that will cause aliasing?

The sampling frequency is $f_s/T$, so the Nyquist frequency is $f_s/2T$. A signal with a temporal frequency component greater than the Nyquist frequency will cause aliasing in the sampled signal.

- 0 points for forgetting the $T$ in $f_s/2T$. (since >90% of you did that, we forgave you)
- 2 points for citing the Nyquist limit, but neglecting to characterize how that relates to the signal being sampled.

4B [5 Points]. In class, we showed a demo of the wagon wheel effect. In this demo, we sampled the wagon wheel, which had 12 spokes, at 24 frames per second – the same speed that a movie camera records frames. When the wagon wheel was moving slowly, everything appeared normal. However, as we sped it up, eventually it appeared to rotate backward. We can characterize the rotational speed of the wagon wheel as $n$ revolutions per minute. Compute the speed at which the wagon wheel will appear to move backward (that is, when aliasing begins to appear).

The wheel is symmetric about a 1/12th turn, so it appears to stand still if it rotates 30° every frame, or 2 revolutions per second. At that speed, the temporal period of oscillation is 1/24th of a second—the sampling rate. Remember the Nyquist limit is two samples per period, or half of that speed, so the wheel will appear to move backward beginning at speeds greater than 1 revolution per second, or 60 revolutions per minute.

- 1 point for forgetting the symmetry of the wheel to arrive at 720 rpm.
- 2 points for forgetting to divide by two, which results in 120 rpm.
4C [5 Points]. We can prevent aliasing by prefiltering the signal before sampling. When we are drawing triangles, we prefilter a triangle using a box filter, which corresponds to computing the coverage of a pixel by the triangle. Now imagine that you have that wagon wheel in physical form in front of you, spinning at a faster speed than you calculated in 4B, and a nice camera with which to take its picture at 24 frames per second. How could you effectively apply a box filter to this signal so that the wagon wheel is not aliased in your movie?

Prefiltering a triangle with a box filter amounts to computing its exact area coverage over a single pixel—the spatial sampling resolution. Likewise, prefiltering the wagon wheel amounts to capturing its appearance over the time period of a single frame, or 1/24th of a second. As you hopefully learned in the homework, you can easily change the shutter speed of a nice camera to set its exposure to 1/24th of a second, or as physically close as it can go while maintaining your frame rate.

While strobing is an interesting idea, it actually gives you the opposite effect of what you want! Deleting intermediate frames is also a bad idea, since you are effectively reducing the sampling rate and thus increasing aliasing. Note that this is a time signal with temporal aliasing, so applying a spatial box filter does not get what you want.

-1 point for realizing you want to average over 1/24s, but not describing how.
-2 points averaging between frames, rather than within a single frame’s time period.

4D [5 Points]. How would the frames of your image sequence look if you applied the box-filtering from 4C, as compared to the those you would get from 4B? (Please provide a little more detail than a one- or two-word description.)

Because you are decreasing the shutter speed, the wheel in your pictures will appear motion-blurred compared to the sharp images from 4B. Specifically, the blur is in a circular direction about the center of the wheel, so that the spokes blend together and the wheel will more like a partially transparent disc rather than having distinct spokes.

Remember that the Nyquist theorem also goes the other way: you don’t get frequencies higher than the Nyquist frequency, so it will never appear as if the wheel were spinning faster (except for the motion blur illusion). Note that it is also aliasing if the wheel appears to turn forward, but at a slower rate than in reality! Because a box filter is not a perfect low-pass filter, you may still see some hints of forward or backward motion, but it will not be as distinct as the 4B sequence.

-3 points for stating the images will be blurry, but failing to characterize how (a spatial blur, though incorrect, will also generate blurry images).
-2 points for observing that the resulting movie will have little or no high temporal frequencies aliased into the sub-Nyquist band, but neglecting to describe what change in appearance the image frames will have.
5. [20 points] GPU Programming

The programmable graphics pipeline allows us to define fragment programs. The fragment program is run once for each fragment drawn. The inputs to the fragment program are the colors interpolated across the triangle and the parameters in the graphics state, including textures. The computed output of the fragment program is stored in the framebuffer.

Suppose you are given a 512 by 512 grayscale texture. We want to write a program using OpenGL fragment shaders to compute the minimum and maximum values in the texture image. Write this program in pseudocode. Give each step of the algorithm, and make sure it is clear that you are using legal OpenGL library calls and legal OpenGL Shading Language constructs. Assume that the graphics system has the following limits:

- No execution of a fragment shader may read more than 128 distinct texel values (the GPU driver will time out if you try to do more work than that).
- You may not have a filter stack deeper than 20 filters in any rendering pass.

The answer we were looking for was an $O(N^2 \log N)$ parallel reduction ($N=512$). There were several ways to accomplish this (varying primarily in the width of the base case in the recursion). Given a base case of 128 texel reads (the max allowed in one shader program execution), a sketch of the filter stack would look like this:

- shader 1: fragments in columns 0, 127, 255, and 383 calculate the min and max of themselves and the 127 texels to their right; store in .r and .g
- shader 2: fragments at col={0,127,255,383} x row={0,127,255,383} do min and max over themselves and the 127 texels below themselves; store in .r and .g
- shader 3: fragment at (0,0) does min and max over the 16 texels written in the last fragment shader pass

Solutions in which all fragments did work (rather than just a subset of them) were not penalized (although they would be dramatically less efficient!); it was also OK to do the min and the max as two passes rather than storing them in vector components in one pass, as long as the total number of shaders in the stack was below 20.
Some of the $O(N^2 \lg N)$ complexity solutions required that the texture size be changed at each shader pass. Since this is more involved than the repeated filtering done in assignment 5, points were taken off unless you made it clear that the subsampling you asked for was possible (i.e., there was OpenGL-ish pseudocode). Several people with this style of solution implemented *incorrect* pseudocode for the downsampling; this was penalized.

There were a few solutions that scaled as $O(N^3)$ but still managed to fit in the constraints listed in the problem spec; these solutions were also accepted. The general flavor of these solutions was that the min/max texels would get bubbled to one or opposite corners in successive filter passes. These solutions typically required a large number of shaders in the stack.

Selected classes of error on this question:
- Many people did not understand the execution model of fragment shaders and their associated data model. In particular fragment shaders, conceptually, all execute **simultaneously** on all fragments in the image. Therefore, any solutions that assumed any serial ordering on the shader executions were incorrect. Additionally, several students tried to do the min/max operation by writing to a shared uniform or varying in the shader. This is incorrect! Uniforms and varyings are both constant data as far as the fragment shader is concerned; even if they were mutable, consider that there is no facility for atomic shared data update in the fragment shader, so the parallelism would break correctness.

- Several people wrote down the sketch of a correct solution, but failed to provide any pseudocode or evidence that what they suggested was implementable in OpenGL. Such solutions received only partial credit.

- A few solutions either did not partition work differently among different fragments (i.e., did no spatial division of labor), or tried to do the whole calculation in one fragment shader pass. This was impossible under the constraints of the problem.

- A few solutions offloaded significant work to the host CPU. This took the form of doing the entire or part of the reduction on the CPU, or work equivalent to this. For example, a few solutions used thresholding and had a "check if all pixels are black" criterion. This requires $O(N^2)$ work on the host to verify and was unacceptable. (Furthermore, the bit depth of the texture was not specified in the question. A HDR texture might be represented as a 32-bit float (aka, FP32 format), in which case you would need ~32 passes to find the min/max by binary search.)
Criteria:
- +20 for a working solution (shader code/pseudocode and enough host code description to convince me that your idea was implementable)
- +18 or +19 for "almost" working solutions that had minor errors (e.g., loop boundary indexing errors in recursion)
- +15 for solutions that were along the right path, but had significant missing content or errors (e.g., incorrect subsampling between shader passes, insufficient pseudocode to convince me that you understood the method, etc.)
- +10 for solutions that got as far as realizing that work had to be partitioned among fragments ("parallelism" - not all frags are doing the same work) and that multiple shader passes would be needed to combine results into a final answer ("recursion")
- +5 for solutions that only had evidence of either parallelism or recursion but not both
- +0 for blank answers, solutions that used the host CPU to do the reduction, solutions that violated the problem constraints, etc.