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 1 1/2 hr. However, you have 3 hours to complete the exam.
1. [20 points] OpenGL.

OpenGL has many geometric primitives, but the fundamental drawing primitive is the triangle. The hardware in the GPU is designed to process triangles very efficiently.

1A [5 points]. OpenGL breaks all polygons (given by a sequence of n vertices inside of glBegin/glEnd) into triangles before drawing them. It does this in a very simple way. How does the OpenGL library break polygons into triangles? Does this method always give the best set of triangles when the polygon is concave? Why or why not?

Sample Solution:
OpenGL breaks polygons into triangle fans.

This method does not always yield the best set of triangles for concave polygons as OpenGL only supports rendering convex polygons for primitive types. In the diagram to the right, the shaded triangle will be filled, but the user expects the bold outline to be filled.

Breakdown of credit:
+ 2 for mentioning or accurately describing triangle fan, +1 for a close answer.
+1 for saying concave will not work well.
+1 to 2 for explanation of why concave fails

1B [5 points]. Triangles are converted to fragments during rasterization. The method used to do this is called the rasterization rule. How does OpenGL determine which fragments to output for each triangle? Briefly describe why this is a good method.

Sample Solution:
OpenGL outputs a fragment if the pixel center lies within the edges of a polygon. This is a good method since it avoids drawing a fragment twice in the case of adjacent polygons, which is bad for performance and will cause visual artifacts with transparent polygons.

Breakdown of credit:
+ 3 for accurately describing rasterization rule.
+1 to 2 for describing why it is a good method.
1C [10 points]. In the past, graphics systems sometimes used quadrilaterals instead of triangles as the fundamental drawing primitive. List two reasons why triangles are a better choice than quadrilaterals. For each reason, describe why triangles work better than quadrilaterals.

Sample Solution:

1) It is easier to perform interpolation with triangles. This is because triangles have 3 vertices which define a plane, while quads have 4 vertices which may not be coplanar, making for difficult interpolation.

2) Triangles are easier to render than quads. Triangles must be convex, while quads can be concave leading to the need to tessellate, or visual artifacts.

Breakdown of credit:
For each reason (5 points each)
+2 to 3 for naming reason triangles are a better choice than quads
+ 2 to 3 for an explanation of what properties triangles have over quads that allow for this behavior

*note: many students explained the properties of triangles that were superior to quads, but failed to mention what advantage these properties had in a graphical environment (eg. Explained that triangles are necessarily coplanar but not an advantage of this, such as easier interpolation).

Alternatively, many students mentioned graphical advantages of triangles such as “faster rendering” but failed to explain what properties of triangles make this the case.

*note: many students didn’t realize that quads could be used to draw triangles by overlapping or aligning vertices. Quad does not mean rectilinear.

*note: We did not give credit for saying that triangles were better because graphics hardware is optimized to draw them – this logic is circular since the optimization is a result of the advantages of using triangles.
[20 points] Transformations

2A [10 points]. Suppose we have a unit triangle (shown on the left) and we want to transform it into the triangle on the right. Write a matrix (give all the entries in the matrix) that transforms the triangle in the desired way.

This question could be solved geometrically or by pure linear algebra. Note that this question must be solved using homogeneous coordinates, as it includes a translation.

**Linear Algebra:**
We are given three points \{(0,0), (1,0), (0,1)\} that must be transformed to \{(1,1), (3,1), (2,2)\}, respectively. This is equivalent to solving for the matrix \(T\) in the following equation:

\[
\begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
T_0 \\
T_1 \\
T_2 \\
\end{bmatrix}
= 
\begin{bmatrix}
1 & 3 & 2 \\
1 & 1 & 2 \\
1 & 1 & 1 \\
\end{bmatrix}
\]

This can be solved as any other linear system.

**Geometry:**
The desired transformation can be composed from three separate transformations: first, scale x by 2; second, shear; third, translate to (1,1).
(The scale can be put elsewhere in the sequence, but it changes the constants in the rest of the transformations.)

\[
S = \begin{bmatrix}
2 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}, \quad H = \begin{bmatrix}
1 & 1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}, \quad X = \begin{bmatrix}
1 & 0 & 1 \\
0 & 1 & 1 \\
0 & 0 & 1 \\
\end{bmatrix}
\]

\(T = XHS\)

Both methods result in the same solution:

\[
T = \begin{bmatrix}
2 & 1 & 1 \\
0 & 1 & 1 \\
0 & 0 & 1 \\
\end{bmatrix}
\]
Grading:
Full credit (10) for correct matrix
-1 for minor algebra errors (copied a coordinate wrong in one step, etc.)
-1 for doing the problem in 4-D homogeneous coordinates rather than 3-D
  (the points are given in 2-D space, not 3-D!)
-2 for incorrect mapping of points (many people put (0,0) in correspondence
  with (2,2), contrary to the given diagram)
-2 if only the component transformations (shear, scale, translate) were given,
  with no final matrix
-2 for providing the transpose of the transformation matrix
-3 for incorrect order of operations converting transformations to the final
  matrix
-3 for a missing operation
-3 for missing homogeneous coordinate
-5 for no matrices given at all
2B [10 points]. Sequence of transformations can often be simplified. Suppose we have the following sequence of 2D transformations: R(90) T(0,1) R(90) T(2,0).

Assume you have a unit square (extending from 0 to 1 in x and y) at the origin. Draw out the intermediate four positions of the square as each transformation is applied.

**Answer presented in world coordinates. It was also OK to use local coordinates, and apply the transformations in the reverse order.**

```
START:

T(2,0)→

R(90)→

T(0,1)→

R(90)→
```

**Grading:**
+4 points for correct illustration, applied right-to-left in world coordinates or left-to-right in local coordinates
-2 for incorrect order of operations with respect to coordinate frame used
-2 for incorrect convention of rotation direction (positive rotations are counterclockwise)
-2 for incorrect application of an operation
-2 for mixing coordinate frames between steps of transformation
-1 for an incomplete drawing
These four transformations can be simplified into one translation followed by one rotation about the origin. What would this translation and rotation be?

\[ R(180)T(3,0) \]

Alternatively, these 4 combined operations could be simplified into one rotation about the origin followed by one translation. What would this rotation and translation be?

\[ T(-3,0)R(180) \]

**Grading:**
+3 on each part if you gave a rotation/translation pair that, when applied in the requested order, resulted in an equivalent drawing to the transformation sketched out in the first part of this question

No credit if specified transformation did not correspond to the correct ending state of your first answer

NB: Many people lost credit here because they specified a transformation that drew a square in the right place, but in the wrong orientation. Orientation matters!
3. [20 points] Interpolation and basis functions

We want to interpolate two points smoothly as shown below. Specifically, we want to find a polynomial that goes through the points \( y=y_0 \) at \( t=0 \) and \( y=y_1 \) at \( t=1 \). Furthermore, we want the polynomial to go through the point \( y=1 \) at \( t=1/2 \) and we want the slope to be 0 at \( t=1/2 \).

3A [4 points]. What degree polynomial should be used?

Because we have four constraints, we should use a cubic polynomial: A polynomial that has degree 3.

\(+4/+4 \text{ for correct solutions. “Degree of freedom = 4” was also accepted.}\)

3B [12 points]. Write out a set of equations that relate the coefficients of the polynomial to the information given as input.

**Parametric form of a cubic polynomial:**

\( P(t) = at^3 + bt^2 + ct + d \)

\( P'(t) = 3at^2 + 2bt + c \)

**Given the four constraints** \( P(0) = y_0, P(1) = y_1, P(1/2) = 1, \) and \( P'(1/2) = 0, \) **here are the 4 equations for computing the coefficients of** \( P(t). \)

\[
\begin{align*}
y_0 &= 0a + 0b + 0c + d \\
y_1 &= 1 \cdot a + 1 \cdot b + 1 \cdot c + d \\
1 &= \frac{1}{8}a + \frac{1}{4}b + \frac{1}{2}c + d \\
0 &= \frac{3}{4}a + b + c
\end{align*}
\]

\(+6/+8/+12 \text{ Students got partial credit for solutions that used wrong polynomials but demonstrated correct reasoning on setting up equations based on the constraints.}\)
3C [4 points]. Solve the equations for the coefficients of the desired polynomial. (This does involve some algebra; we recommend that you do this part after you finish the rest of the exam).

\[
\begin{bmatrix}
  y_0 \\
  y_1 \\
  1 \\
  0
\end{bmatrix}
= 
\begin{bmatrix}
  0 & 0 & 0 & 1 & a \\
  1 & 1 & 1 & 1 & b \\
  1/8 & 1/4 & 1/2 & 1 & c \\
  3/4 & 1 & 1 & 0 & d
\end{bmatrix}
\]

\[
\begin{bmatrix}
  a \\
  b \\
  c \\
  d
\end{bmatrix}
= 
\begin{bmatrix}
  4y_1 - 4y_0 \\
  8y_0 - 4y_1 - 4 \\
  y_1 - 5y_0 + 4 \\
  y_0
\end{bmatrix}
\]

+3/+4 for small errors.
+2/+4 for students who gave wrong answers but demonstrated understanding of how to solve 4 equations with 4 unknowns.
4. [20 points] Bezier Curve

The most widely used curve in computer graphics is the Bezier curve. A cubic Bezier curve is determined by 4 control points $P_0, P_1, P_2, \text{ and } P_3$.

![Bezier Curve Diagram]

A natural extension of your morphing assignment would be to use Bezier curves instead of line segments for features. Instead of drawing two corresponding edges, the artist would draw two corresponding Bezier curves.

The warp used in the morph is based on computing the distance from a point in the image to the line segment. So, if you could compute the distance from a point $P$ to a Bezier curve $(P_0, P_1, P_2, P_3)$ you could use the same formula as you used for line segments.

How would you compute the distance from a point to a Bezier curve? Describe your algorithm in detail using pseudocode. Make sure to write out the math. [Hint: You may still want to use the function you wrote in the morphing assignment that computes the distance from a point to a line segment.]

A good answer for this problem is to keep subdividing the Bezier curve until the points on the curve are very close to being collinear, then calculating the distance from $P$ to the lines, and returning the minimum for each subdivision.

\[
\text{BezDist}(P, B = \{P_0, P_1, P_2, P_3\}) \\
\quad \text{if } \text{Dist}(P_1, P_0, P_3) < \epsilon \text{ AND } \text{Dist}(P_2, P_0, P_3) < \epsilon \text{ // collinear?} \\
\quad \quad \text{return } \text{Dist}(P, P_0, P_3) \\
\quad \text{else} \\
\quad \quad (L,R) = \text{Subdivide}(P_0, P_1, P_2, P_3) \\
\quad \quad \text{return } \min(\text{BezDist}(P,L), \text{BezDist}(P,R))
\]

You could optimize this a bit by keeping track of the distance between the convex hulls of the Bezier curves and the point $P$. You can use a priority queue to always subdivide the closest curve, until you’ve found a line which is closer than any other convex hull, and must give the correct answer.
A less efficient implementation would subdivide until all the curves until the endpoints are within a pixel, rather than until the points are collinear. This would be deducted one point.

Brute force techniques, such as checking every t along the line and returning the minimum one, were deducted 2 points. Note that you can’t break out of the loop once you’ve found a local minimum, as the Bezier curve can have several minima if it has a loop in it.

Some students tried to solve for roots of the equation \( B'(t)(P-B(t))=0 \). This is a 5th degree polynomial and it is not trivial to solve. We cannot use bisection, golden section search, etc, because the equation is not monotonic if there are loops in the curve. These techniques usually resulted in at least -5.
5. [20 points] Input and Geometry

Your friend Brittany has an idea for a new input device. She was watching the Superbowl and noticed the camera that flies around the stadium on cables. She sketches out the following design for a 2D table-top input device based on similar principles:

In her design, there is a “puck” attached to three cables. Each cable is attached to a pulley on a post. As the puck moves around in 2D, the cables wind or unwind around the pulley. Each pulley is attached to a spring to ensure that the cable is always taut (that is there is no slack in the cable). In this design, the posts are at (0,0), (1,0), and (0,1). When you heard her idea, you raced to implement it.

5A [10 points] How could you measure the length of each cable? That is, what type of sensing could be used to keep track of the length of the cable?

You can apply what you learned from the input technology lecture and mount rotary encoders with quadrature encoding on each of the pulleys. As each cable winds around (or onto) its pulley, the +/- change in encoder counts can tell you how much cable was pulled in or let out, as follows:

\[
\Delta L = \frac{\Delta \text{counts} \times 2\pi r}{\text{counts per rev.}}
\]

where \( r \) is the radius of the pulley.

This assumes that the cables are parallel to the table surface, are wound evenly on the pulley, and that there is no slip in the system. With just a little bit of good engineering, this technique can be made to measure the cable lengths very accurately.

Common deductions:
- up to -5 for having the right idea, but neglecting to discuss the type of sensor to use or where to put them to obtain the needed information
- up to -2 for describing a method that may work, but is difficult to achieve the accuracy needed or is prohibitively expensive for this application
5B [10 points] Given the positions of the posts (see above) and the lengths of the cables connecting each post to the puck (a, b and c), how would you compute the position (x, y) of the puck?

Let \( P = (x, y) \) be the position of the puck, and \( P_0 = (0,0), P_1 = (1,0), P_2 = (0,1) \) be the positions of the three posts. The most straightforward way to solve this part is to apply the distance formula or Pythagorean theorem from the posts to the point \( P \). You then obtain three equations:

\[
\begin{align*}
a^2 &= x^2 + y^2 \\
b^2 &= (1 - x)^2 + y^2 = 1 - 2x + x^2 + y^2 \\
c^2 &= (1 - y)^2 + x^2 = 1 - 2y + y^2 + x^2
\end{align*}
\]

Substituting \( a^2 \) for \( x^2 + y^2 \) in the second and third equations, then simplifying, gives you the final result:

\[
P = (x, y) = \left( \frac{1 + a^2 - b^2}{2}, \frac{1 + a^2 - c^2}{2} \right)
\]

You may also notice that \( x \) is the height perpendicular to \( P_0P_2 \) and \( y \) is height perpendicular to \( P_0P_1 \). Thus, you can also arrive at \((x, y)\) using the triangle area \( A = \frac{1}{2}bh \) (with \( b = 1 \)) combined with Heron’s formula as follows:

\[
\begin{align*}
x &= 2A_1 = \sqrt{s(s-a)(s-1)(s-c)}, \quad \text{where } s = \frac{1}{2}(a + 1 + c) \\
y &= 2A_2 = \sqrt{s(s-a)(s-b)(s-1)}, \quad \text{where } s = \frac{1}{2}(a + b + 1)
\end{align*}
\]

Finally, a long, but popular way to arrive at the answer is to use barycentric interpolation of the Cartesian coordinates at the posts. You would calculate weights and write \( P \) as follows (using Heron’s formula again for area):

\[
\begin{align*}
\alpha &= \frac{A(PP_1P_2)}{A(P_0P_1P_2)}, \quad \beta = \frac{A(PP_2P_0)}{A(P_0P_1P_2)}, \quad \gamma = \frac{A(PP_0P_1)}{A(P_0P_1P_2)} \\
P &= \alpha P_0 + \beta P_1 + \gamma P_2
\end{align*}
\]

If you use this method, you should convince yourself (and us) that it works for the entire table, and not just the triangular part bounded by the three posts.

Common deductions:
- up to -3 if you didn’t manage to write \((x, y)\) as a formula in terms of \( a, b, \) and \( c \) for computing the location of the puck
- up to -6 if you started with a reasonable idea, but weren’t able to carry it through to completion