CS 130, Final

Solutions

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Read the entire exam before beginning. **Manage your time carefully.** This exam has 60 points; you need 50 to get full credit.

Problem 1 (2 points)

Let $q(\theta, \vec{u}) = (\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\vec{u})$ be the rotation about axis $\vec{u}$ ($\|\vec{u}\| = 1$) by angle θ. Show that $q(\phi, \vec{u})q(\theta, \vec{u}) = q(\phi + \theta, \vec{u})$.

$$
\begin{aligned}
q(\phi, \vec{u})q(\theta, \vec{u}) &= (\cos\tfrac{\phi}{2}, \sin\tfrac{\phi}{2}\vec{u})(\cos\tfrac{\theta}{2}, \sin\tfrac{\theta}{2}\vec{u}) \\
&= (\cos\tfrac{\phi}{2}\cos\tfrac{\theta}{2} - \sin\tfrac{\phi}{2}\sin\tfrac{\theta}{2}\vec{u}\cdot\vec{u}, \cos\tfrac{\phi}{2}\sin\tfrac{\theta}{2}\vec{u} + \cos\tfrac{\theta}{2}\sin\tfrac{\phi}{2}\vec{u} + \sin\tfrac{\phi}{2}\sin\tfrac{\theta}{2}\vec{u}\times\vec{u}) \\
&= (\cos\tfrac{\phi}{2}\cos\tfrac{\theta}{2} - \sin\tfrac{\phi}{2}\sin\tfrac{\theta}{2}, (\cos\tfrac{\phi}{2}\sin\tfrac{\theta}{2} + \cos\tfrac{\theta}{2}\sin\tfrac{\phi}{2})\vec{u}) \\
&= (\cos(\tfrac{\phi}{2} + \tfrac{\theta}{2}), \sin(\tfrac{\phi}{2} + \tfrac{\theta}{2})\vec{u}) \\
&= q(\phi + \theta, \vec{u})
\end{aligned}
$$

Problem 2 (1 points)

If I were interviewing a programming candidate, here is a question I might ask. Rewrite the function `double foo(double x){return pow(x,15);}` by replacing `pow` with at most 6 multiplications. (Fewer than 6 multiplications gives you one point of extra credit.) Division is not allowed.

You can get to 6 by eliminating common subexpressions.

```
double foo(double x){return pow(x,15);}
double foo(double x){return x*x*x*x*x*x*x*x*x*x*x*x*x*x*x;}
double foo(double x){double a=x*x;return a*a*a*a*a*a*a*x;}
double foo(double x){double a=x*x,b=a*a;return b*b*b*a*x;}
```

You can get the result in 5 multiplications by noting that $3 \cdot 5 = 15$.

```
double foo(double x){double a=x*x,b=a*a*x;return b*b*b;}
```

Problem 3 (2 points)

Eliminate multiplications from the body of the loop.

```
void func(double a, double b, int n)
{
  for(int i=-n; i<n; i++)
  {
    foo(a * b + a * i - b * i)
  }
}
```

```
void func(double a, double b, int n)
{
  double d = a - b, c = a * b - n * d;
  for(int i=-n; i<n; i++)
  {
    foo(c);
    c += d;
  }
}
```

Problem 4 (2 points)

Eliminate `exp` from the body of the loop.

```
void func(double a, double b, int n)
{
  for(int i=0; i<n; i++)
  {
    foo(exp(a - b * i))
  }
}
```

Let $c_i = e^{a-bi}$. Then, $c_i = e^{-b}c_{i-1}$ and $c_0 = e^a$.

```
void func(double a, double b, int n)
{
  double c = exp(a), d = exp(-b);
  for(int i=0; i<n; i++)
  {
    foo(c);
    c *= d;
  }
}
```

Problem 5 (2 points)

Eliminate the trig functions from the body of the loop.

```
void func(double a, double b, int n)
{
  for(int i=0; i<n; i++)
  {
    foo(cos(i*2*pi/n),sin(i*2*pi/n));
  }
}
```

Let $c_i = \cos(\frac{2i\pi}{n})$ and $s_i = \sin(\frac{2i\pi}{n})$. Note that the angle addition formulas imply $c_{i+k} = c_i c_k - s_i s_k$ and $s_{i+k} = s_i c_k + c_i s_k$. We will use these with $k = 1$.

```
void func(double a, double b, int n)
{
  double dt = 2*pi/n, c1 = cos(dt), s1 = sin(dt), c = 1, s = 0;
  for(int i=0; i<n; i++)
  {
    foo(c,s);
    double tc = c * c1 - s * s1; // Do not destroy c before computing s!
    s = s * c1 + c * s1;
    c = tc;
  }
}
```

Problem 6 (4 points)

Write pseudocode to rasterize the parabola $4py = x^2$ in the box $[-2pa, 2pa] \times [0, pa^2]$, where a and p are positive integers that are inputs to your function `void rasterize(int a,int p)`. The line must be thin and without gaps. No floating point is allowed, but basic arithmetic (add, subtract, multiply, divide) is permitted everywhere. Use the routine `draw(int x,int y)` to plot points; this routine can be called with any integer coordinates inside the box. Your algorithm should be an extension of the midpoint algorithm.

This problem is fairly similar to ones we have done before. One tricky part is that we need to pay attention to when $|y'| \le 1$. Note that $y = \frac{x^2}{4p}$, so $y' = \frac{x}{2p}$. $|y'| \le 1$ when $|x| \le 2p$. For these x, we must increment by x. Beyond that, the loop should run in y. We should also be careful to not go beyond $|x| = 2pa$ or $y = pa^2$. This is made somewhat easier by the nice rectangle (the parabola hits both top corners). Note that $2pa \ge pa$, so the first loop never hits the edge.

```
void rasterize(int a, int p)
{
  int x = 0, y = 0, mx = 2*p, my = p*a*a;
  draw(x,y);
  while(x < mx)
  {
    x++;
    if(p*(4*y+2) < x*x)
        y++;
    draw(-x,y);
    draw(x,y);
  }
  while(y < my)
  {
    y++;
    if(16*p*y > (2*x+1)*(2*x+1))
        x++;
    draw(-x,y);
    draw(x,y);
  }
}
```

Note that I chose to increment the primary direction *before* computing the criterion to produce simpler code.

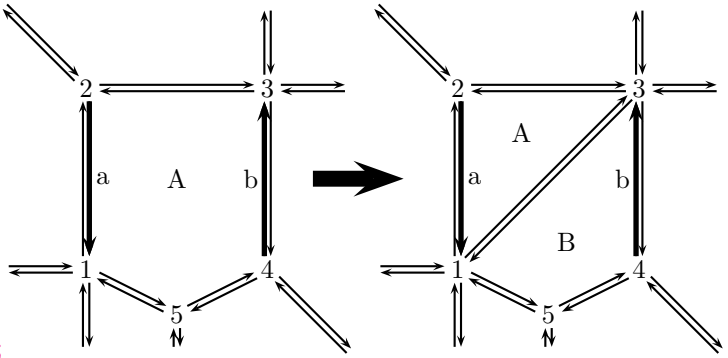
For the first part of the midpoint algorithm, we need to test the point $(x, y+\frac{1}{2})$, noting that x has already been incremented. We increment y if $4p(y+\frac{1}{2}) < x^2$.

For the second part of the midpoint algorithm, we need to test the point $(x+\frac{1}{2}, y)$, noting that y has already been incremented. We increment x if $4py > (x+\frac{1}{2})^2$. Clearing fractions gives the criterion in the code.

Problem 7 (4 points)

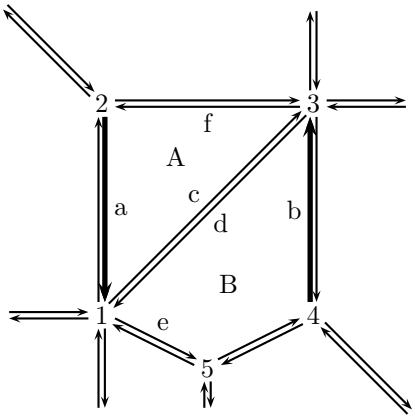
For this problem, you will write the routine `void Split_Face(coedge* a, coedge* b);`. The coedges `a` and `b` are coedges belonging to the same face (call it `A`). That is, `a->cell==b->cell`. You may assume `a` and `b` do not share an endpoint. These coedges point to two vertices of face `A` (1 and 3 in this case.) You will connect these two points to split the face in two, as shown below. You will need to allocate a new `face` (call it `B`) and two new `coedge`s. After your routine is finished: (1) the half-edge data structure is in a valid state, (2) the face that was `A` is now divided into two faces `A` and `B`, (3) `a->cell==A`, (4) `b->cell==B`. The C++ structures used for `vertex`, `face`, and `coedge` are shown below.

```
struct vertex
{
  coedge* edge;
};
struct face
{
  coedge* edge;
};
struct coedge
{
  coedge *pair, *next, *prev;
  vertex *head, *tail;
  face *cell;
};
```



The strategy I take is to first create the new objects, then go through the different structure members one by one and make sure I have them all updated. Note that none of the `vertex->edge` entries need to change. The named quantities in the routine are shown in the diagram.

```
void Split_Face(coedge* a, coedge* b)
{
  face* A = a->cell;
  face* B = new face;
  coedge* c = new coedge;
  coedge* d = new coedge;
  coedge* e = a->next;
  coedge* f = b->next;
  A->edge = a;
  B->edge = b;
  c->pair = d; d->pair = c;
  a->next = c; c->prev = a;
  c->next = f; f->prev = c;
  b->next = d; d->prev = b;
  d->next = e; e->prev = d;
  c->head = b->head;
  c->tail = a->head;
  d->head = a->head;
  d->tail = b->head;
  c->cell = A;
  d->cell = B;
  for(coedge* p = d->next; p != d; p = p->next)
    p->cell = B;
}
```



Problem 8 (4 points)

Design an algorithm for implementing a routine `bool clip(vec2 p, vec2 n, vector<vec2>& poly);`.

- Here, `poly` is a list of vertices of a convex polygon (listed in counterclockwise order).
- The vertex that is listed first in `poly` is arbitrary.
- The vectors `p` and `n` define a line. A point $\vec{x}$ considered to be inside if $(\vec{x} - \vec{p}) \cdot \vec{n} \leq 0$.
- The clipping line intersects the polygon exactly zero or two times
- The clipping line never passes through a vertex of the polygon.

The program should have the following behavior:

- If the polygon is entirely inside, `poly` is not modified, return `false`. Otherwise, return `true`.
- If the polygon lies entirely outside, `poly` should be empty when the function returns.
- Otherwise, the polygon `poly` should be a clipped version of the original.

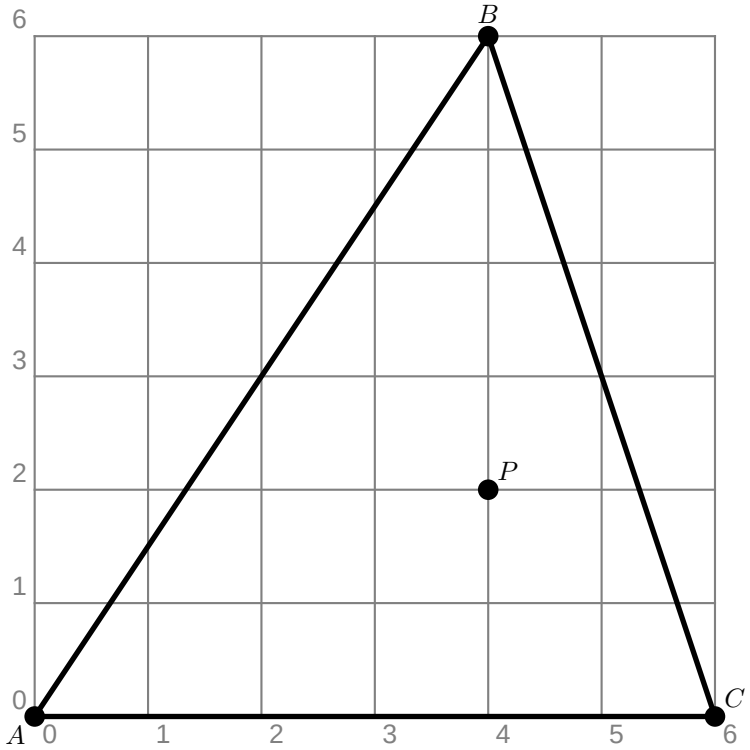
I do not expect you to write code or pseudocode for this, but you should explain the algorithm in enough detail that someone could sit down and implement it. You may assume that the person implementing the routine knows how to find the intersection between two lines.

There are two special cases that need to be handled separately: the polygon is entirely inside or entirely outside. If neither is the case, we want to find the edge where the polygon enters the clipping region (the assumptions ensure there will be exactly one). Below is an algorithm that will work.

1. Walk through the polygon testing each edge to find the edge that crosses from outside to inside (enter edge) and the edge that crosses from inside to outside (exit edge). Don't forget the segment between the first and last entries of `poly`. You can determine whether such edges were found by initializing them to an invalid index and testing for this value later.
2. If no such edges were found, the polygon is entirely inside or entirely outside. Test the first vertex. If it is inside, the polygon is entirely inside; return `false`. Otherwise, clear out `poly` and return `true`.
3. Create a temporary polygon `vector<vec2> temp`.
4. Append the intersection point between the enter edge and the clip line.
5. Starting with the inside endpoint of the enter edge, walk through the polygon list until the inside endpoint of the exit edge is encountered, adding the vertices to `temp` as they are encountered (including both of those endpoints). Note that if the exit edge index is less than the enter edge index, you will pass the end of `poly` and have to resume from the beginning.
6. Append the intersection point between the exit edge and the clip line.
7. Swap `temp` with `poly` and return `true`.

Problem 9 (2 points)

The triangle at right is to be rasterized. The colors of the vertices are A = yellow = $(1, 1, 0)$, B = cyan = $(0, 1, 1)$ and, C = violet = $(1, 0, 1)$. Compute the color of the point P.

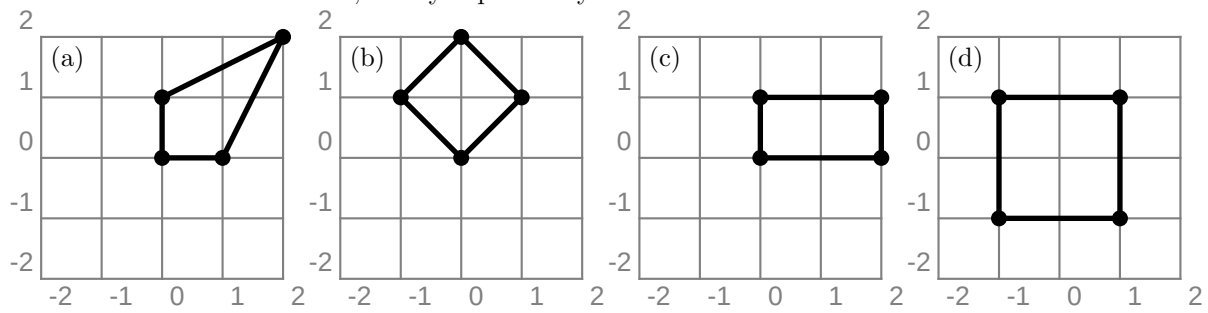


To do this, we need to compute the areas of the triangles, which we can then use to compute barycentric weights.

$$\text{area}(ABC) = 18$$
$$\text{area}(PBC) = 4$$
$$\text{area}(APC) = 6$$
$$\text{area}(ABP) = 8$$
$$\alpha = \frac{\text{area}(PBC)}{\text{area}(ABC)} = \frac{2}{9}$$
$$\beta = \frac{\text{area}(APC)}{\text{area}(ABC)} = \frac{3}{9}$$
$$\gamma = \frac{\text{area}(ABP)}{\text{area}(ABC)} = \frac{4}{9}$$
$$C_P = \alpha C_A + \beta C_B + \gamma C_C = \left(\frac{6}{9}, \frac{5}{9}, \frac{7}{9}\right)$$

Problem 10 (4 points)

The unit square $[0,1] \times [0,1]$ is transformed to each of the polygons below. In each case, identify the type of transform and, if possible, find a 3×3 homogeneous transform matrix corresponding to it. (R=rotation, T=translation, S=uniform scale, A=affine, L=linear, P=projection, X=none of these. R, S, and T can be combined.) The most restrictive option should be chosen. Thus, a transform that can be accomplished by a combination of rotation and uniform scale should be described as R+S, not as A. If no transform matrix exists, briefly explain why.



(a) P. To find it, assume the matrix is $\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$.

First point: $\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ k \end{pmatrix}$. Implies $c = 0$, $f = 0$.

Second point: $\begin{pmatrix} a & b & 0 \\ d & e & 0 \\ g & h & i \end{pmatrix}\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} k \\ 0 \\ k \end{pmatrix}$. Implies $d = 0$, $a = g + i$.

Third point: $\begin{pmatrix} g+i & b & 0 \\ 0 & e & 0 \\ g & h & i \end{pmatrix}\begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ k \\ k \end{pmatrix}$. Implies $b = 0$, $e = h + i$.

Fourth point: $\begin{pmatrix} g+i & 0 & 0 \\ 0 & h+i & 0 \\ g & h & i \end{pmatrix}\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2k \\ 2k \\ k \end{pmatrix}$. $2k = g + i = h + i$. $k = g + h + i$. $g = h = -k$. $i = 3k$.

Since the overall scale does not matter, let $k = 1$. Solution: $\begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ -1 & -1 & 3 \end{pmatrix}$.

(b) RS. $\begin{pmatrix} 1 & -1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

(c) L. $\begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

(d) ST. $\begin{pmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{pmatrix}$

Problem 11 (4 points)

A box is given by $[a,b] \times [c,d] \times [e,f]$. Let $\vec{u} + t\vec{w}$ define a ray ($\|\vec{w}\| = 1$). Work out logic for a routine that computes the intersections. The routine should produce a list of all suitable intersection pairs as was done for the second project. I am mostly concerned with the logic and the algebra. Just be sure it is clear what pairs you produce in each case. You must handle all cases to get full credit. Don't worry about being efficient or about degeneracies (division by zero, etc.). You can get most of the points by solving the 2D version.

The plane $x = a$ is crossed when $u_x + tv_x = a$, or $t_a = \frac{a-u_x}{v_x}$. Similarly, $t_b = \frac{b-u_x}{v_x}$. Let $t^x_- = \min(t_a, t_b)$ and $t^x_+ = \max(t_a, t_b)$. The line (not ray) enters the space between the x faces at t^x_- and leaves at t^x_+. This gives us the interval $[t^x_-, t^x_+]$. Repeating this process in the y and z directions we get $[t^y_-, t^y_+]$ and $[t^z_-, t^z_+]$. We must intersect these intervals.

Let $A = \max(t^x_-, t^y_-, t^z_-)$. This is the time when the last entering planes has been crossed. $B = \min(t^x_+, t^y_+, t^z_+)$ is the time when the first leaving plane is encountered. If $A > B$, there is no intersection. Otherwise, the intersection of the intervals is $[A, B]$.

Now, we have to take account of the fact that part of the interval may be behind the ray. If $0 \le A \le B$, output (A, B). If $A < 0 \le B$ output $(0, B)$. Otherwise $A \le B < 0$, and there is no intersection.

Problem 12 (1 points)

For what values of `x,y,z` is the code fragment

```
glBegin(GL_POINTS);
glVertex3f(x,y,z);
glEnd();
```

equivalent to this fragment?

```
glPushMatrix();
glRotatef(180,0,0,1);
glTranslatef(0,2,1);
glScalef(-1,1,1);
glBegin(GL_POINTS);
glVertex3f(1,2,3);
glEnd();
glPopMatrix();
```

Recall that the operations are specified to OpenGL in reverse order.

$$\begin{pmatrix}1\\2\\3\end{pmatrix} \xrightarrow{\text{scale}} \begin{pmatrix}-1\\2\\3\end{pmatrix} \xrightarrow{\text{translate}} \begin{pmatrix}-1\\4\\4\end{pmatrix} \xrightarrow{\text{rotate}} \begin{pmatrix}1\\-4\\4\end{pmatrix}$$

Thus, $x = 1$, $y = -4$, and $z = 4$.

Problem 13 (4 points)

A conic section (in 2D) can be defined by a 3×3 symmetric homogeneous matrix $\mathbf{A}$. A homogeneous point $\vec{w} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ is on the conic section if and only if $\vec{w}^T \mathbf{A} \vec{w} = 0$. Observe that scaling $\vec{w}$ does not affect the actual 2D point or whether it is considered to be on the conic section.

(a) Find $\mathbf{A}$ corresponding to a general circle: $(x-h)^2 + (y-k)^2 = r^2$.

(b) Show that an ellipse $\frac{(x-h)^2}{m^2} + \frac{(y-k)^2}{n^2} = 1$ and a hyperbola $\frac{(x-h)^2}{m^2} - \frac{(y-k)^2}{n^2} = 1$ can both be written in the form $ax^2 + 2bxy + cy^2 + 2dx + 2ey + f = 0$.

(c) Show that things of the form $ax^2 + 2bxy + cy^2 + 2dx + 2ey + f = 0$ are exactly what can be expressed by $\vec{w}^T \mathbf{A} \vec{w} = 0$. Find $\mathbf{A}$.

(d) Use this form of a conic section to show that conic sections are preserved under projective transforms. (Assume the transform is invertible.)

(a) $\mathbf{A} = \begin{pmatrix} 1 & 0 & -h \\ 0 & 1 & -k \\ -h & -k & h^2 + k^2 - r^2 \end{pmatrix}$

(b) Ellipse: $a = \frac{1}{m^2}$, $b = 0$, $c = \frac{1}{n^2}$, $d = -\frac{h}{m^2}$, $e = -\frac{k}{n^2}$, $f = \frac{h^2}{m^2} + \frac{k^2}{n^2} - 1$.

Hyperbola: $a = \frac{1}{m^2}$, $b = 0$, $c = -\frac{1}{n^2}$, $d = -\frac{h}{m^2}$, $e = \frac{k}{n^2}$, $f = \frac{h^2}{m^2} - \frac{k^2}{n^2} - 1$.

(c) $\mathbf{A} = \begin{pmatrix} a & b & d \\ b & c & e \\ d & e & f \end{pmatrix}$. $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}^T \begin{pmatrix} a & b & d \\ b & c & e \\ d & e & f \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = ax^2 + 2bxy + cy^2 + 2dx + 2ey + f = 0$. The two are equivalent.

(d) Let $\mathbf{P}$ be a matrix representing a projective transform. Let $\mathbf{A}$ be a symmetric matrix representing a conic section. A homogeneous point $\vec{w}$ on this conic section is transformed to $\vec{u} = \mathbf{P}\vec{w}$.

$$
\begin{aligned}
0 &= \vec{w}^T \mathbf{A} \vec{w} \\
0 &= \vec{w}^T \mathbf{P}^T \mathbf{P}^{-T} \mathbf{A} \mathbf{P}^{-1} \mathbf{P} \vec{w} \\
0 &= \vec{u}^T (\mathbf{P}^{-T} \mathbf{A} \mathbf{P}^{-1}) \vec{u} \\
0 &= \vec{u}^T \mathbf{B} \vec{u}
\end{aligned}
$$

Thus, we see that $\vec{u}$ is on a conic section defined by the symmetric matrix $\mathbf{B} = \mathbf{P}^{-T} \mathbf{A} \mathbf{P}^{-1}$, which defines another conic section.

Problem 14 (4 points)

To simplify some computations, we introduced the notation $\vec{u}^*$ to represent the matrix such that $\vec{u}^*\vec{v} = \vec{u} \times \vec{v}$. We can do something similar for quaternions. Let $p = (r, \vec{u})$, $q = (s, \vec{v})$, and $pq = (t, \vec{w})$. Define $q^\triangle = \begin{pmatrix} s \\ \vec{v} \end{pmatrix}$ and $q^\square = \begin{pmatrix} s & -\vec{v}^T \\ \vec{v} & s\mathbf{I} + \vec{v}^* \end{pmatrix}$. Note that $q^\triangle$ is a 4-vector and $q^\square$ is a 4×4 matrix.

(a) Show that $p^\square q^\triangle = (pq)^\triangle$.

(b) What is $1^\square$?

(c) Show that $p^\square q^\square = (pq)^\square$ by multiplying the matrices. You may find these properties of cross product matrices useful: $(\vec{u} \times \vec{v})^* = \vec{v}\vec{u}^T - \vec{u}\vec{v}^T$, $\vec{u}^*\vec{v}^* = \vec{v}\vec{u}^T - (\vec{u} \cdot \vec{v})\mathbf{I}$.

(d) Let m be another quaternion. Use the new tools introduced above to show that $(mp)q = m(pq)$. That is, quaternion multiplication is associative. Be careful not to assume the conclusion in the process. (Hint: your solution should be quite short.)

Recall that $pq = (rs - \vec{u} \cdot \vec{v}, s\vec{u} + r\vec{v} + \vec{u} \times \vec{v})$.

(a) $p^\square q^\triangle = \begin{pmatrix} r & -\vec{u}^T \\ \vec{u} & r\mathbf{I} + \vec{u}^* \end{pmatrix} \begin{pmatrix} s \\ \vec{v} \end{pmatrix} = \begin{pmatrix} rs - \vec{u}^T\vec{v} \\ s\vec{u} + r\vec{v} + \vec{u}^*\vec{v} \end{pmatrix} = (pq)^\triangle$.

(b) $1^\square = \mathbf{I}$.

(c)

$$
\begin{aligned}
p^\square q^\square &= \begin{pmatrix} r & -\vec{u}^T \\ \vec{u} & r\mathbf{I} + \vec{u}^* \end{pmatrix} \begin{pmatrix} s & -\vec{v}^T \\ \vec{v} & s\mathbf{I} + \vec{v}^* \end{pmatrix} \\
&= \begin{pmatrix} rs - \vec{u}^T\vec{v} & -r\vec{v}^T - s\vec{u}^T - \vec{u}^T\vec{v}^* \\ s\vec{u} + r\vec{v} + \vec{u}^*\vec{v} & -\vec{u}\vec{v}^T + (r\mathbf{I} + \vec{u}^*)(s\mathbf{I} + \vec{v}^*) \end{pmatrix} \\
&= \begin{pmatrix} t & -r\vec{v}^T - s\vec{u}^T - ((\vec{v}^*)^T\vec{u})^T \\ \vec{w} & -\vec{u}\vec{v}^T + rs\mathbf{I} + s\vec{u}^* + r\vec{v}^* + \vec{u}^*\vec{v}^* \end{pmatrix} \\
&= \begin{pmatrix} t & -r\vec{v}^T - s\vec{u}^T + (\vec{v}^*\vec{u})^T \\ \vec{w} & -\vec{u}\vec{v}^T + rs\mathbf{I} + s\vec{u}^* + r\vec{v}^* + \vec{v}\vec{u}^T - \vec{u}^T\vec{v}\mathbf{I} \end{pmatrix} \\
&= \begin{pmatrix} t & -r\vec{v}^T - s\vec{u}^T + (\vec{v} \times \vec{u})^T \\ \vec{w} & t\mathbf{I} + s\vec{u}^* + r\vec{v}^* + (\vec{u} \times \vec{v})^* \end{pmatrix} \\
&= \begin{pmatrix} t & -\vec{w}^T \\ \vec{w} & t\mathbf{I} + \vec{w}^* \end{pmatrix} \\
&= (pq)^\square
\end{aligned}
$$

(d) First, $((mp)q)^\triangle = (mp)^\square q^\triangle = (m^\square p^\square)q^\triangle = m^\square(p^\square q^\triangle) = m^\square(pq)^\triangle = (m(pq))^\triangle$. Note that I have only used associativity for matrices and vectors here. Finally, note that if $p^\triangle = q^\triangle$ then $p = q$. Alternatively, one could use matrices throughout. $((mp)q)^\square = (mp)^\square q^\square = (m^\square p^\square)q^\square = m^\square(p^\square q^\square) = m^\square(pq)^\square = (m(pq))^\square$. As before, if $p^\square = q^\square$ then $p = q$.

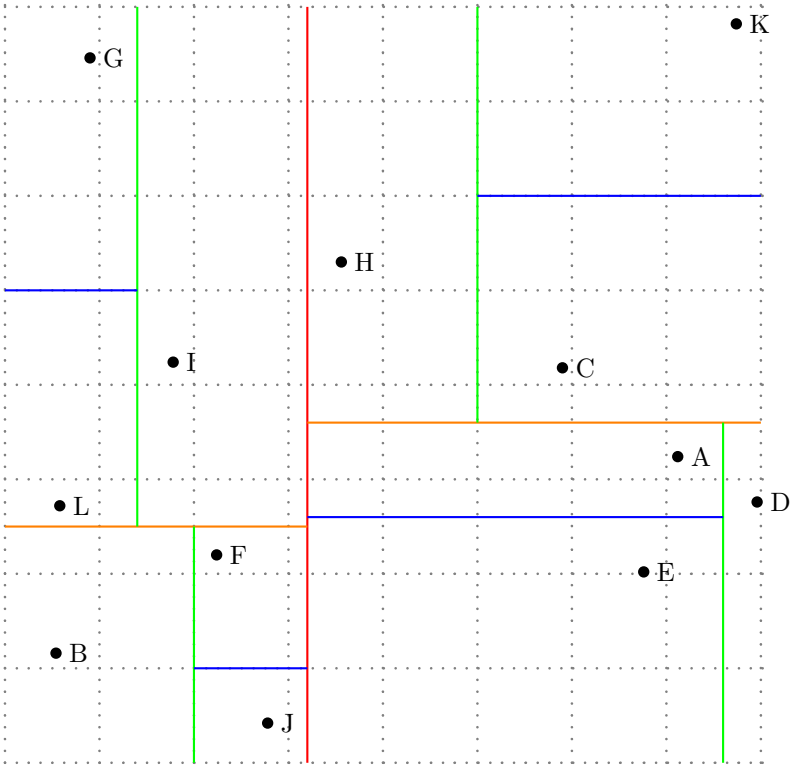
Problem 15 (4 points)

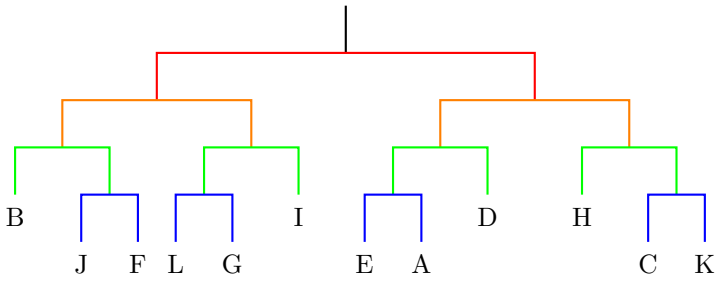
Raytracing traces photons back from the camera to determine the color of pixels in an image. The process is fairly efficient, but it has serious limitations.
(a) When introducing ray tracing, we assumed that light hitting an object arrived at the object straight from the light (we also did an ambient hack). If this straight line is blocked by an object, we assume that the surface receives no photons from the light source. In reality, light can reach a surface by reflecting from a mirror or refracting through a glass object. Why do we ignore this light?
(b) An alternative strategy is to follow the photons forward from the lights rather than backwards from the camera. The image is made when photons hit the film as they would in a real camera. Why would this strategy work in the above scenarios?
(c) This method is not used in practice because it is too slow. Why is this approach vastly slower than normal ray tracing?
(d) A more practical approach is to shoot photons and record where they hit objects in the scene. Then, we make a texture map for each object showing where and how many photons hit the object. Finally, we ray trace the image as we normally would. This is the general idea behind photon mapping. Why would this be much faster than the original approach from (b)?

(a) There is not generally a good way to determine where to cast a ray to see if the light is visible. It is easy enough with a flat mirror, but things get hard in a hurry. To get one bounce with one light, you would have to consider each transparent or reflective object and solve for all possible ray directions from which light could travel through the object to reach the camera. This is a very hard (and nonlinear) problem. We don't do it because we don't have a good way to do so.
(b) It is able to capture physics because it is mimicking the physics directly. There is no problem tracing a ray through transparent objects; the path of the ray is easy to compute.
(c) In normal ray tracing, pretty much every ray we cast counts. Other than shadow rays, all rays we cast contribute to the image. In the above scheme, only rays that happen to strike our tiny piece of film will contribute to the output. The vast majority hit other surfaces.
(d) Photons illuminate objects by hitting them. We don't have to get lucky by having a photon be re-emitted from the surface and hit the film.

Problem 16 (4 points)

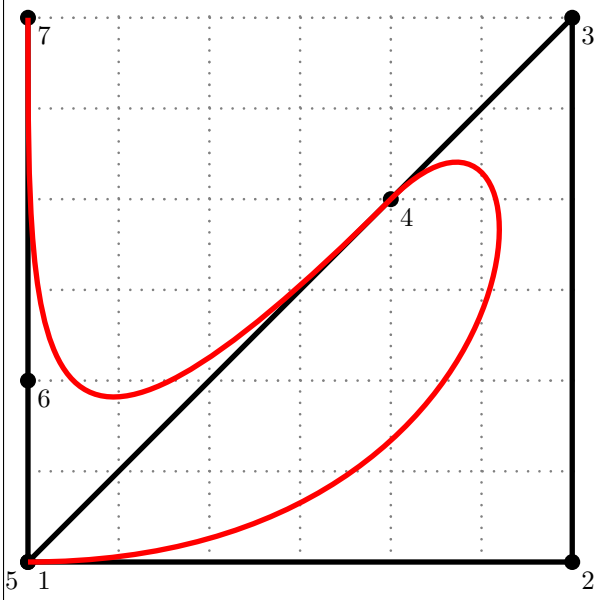
Construct a KD-tree (we also called this a BSP in class) for the following points. Construct the actual tree.

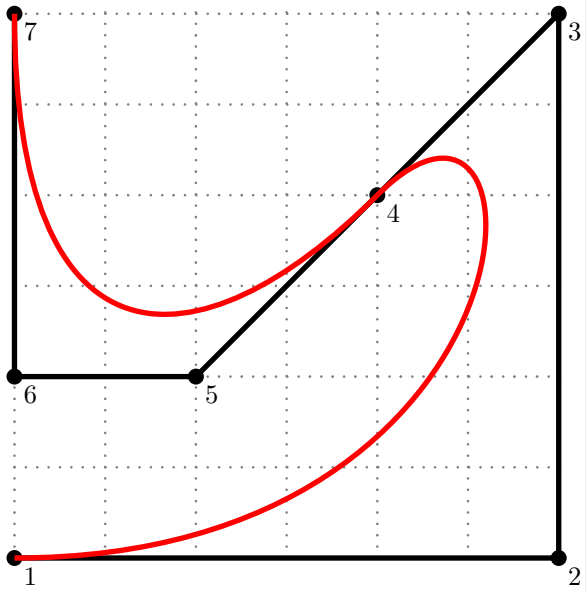


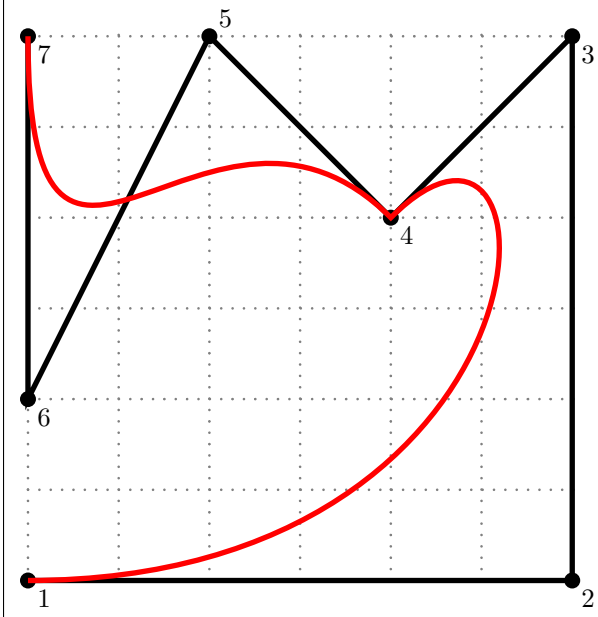


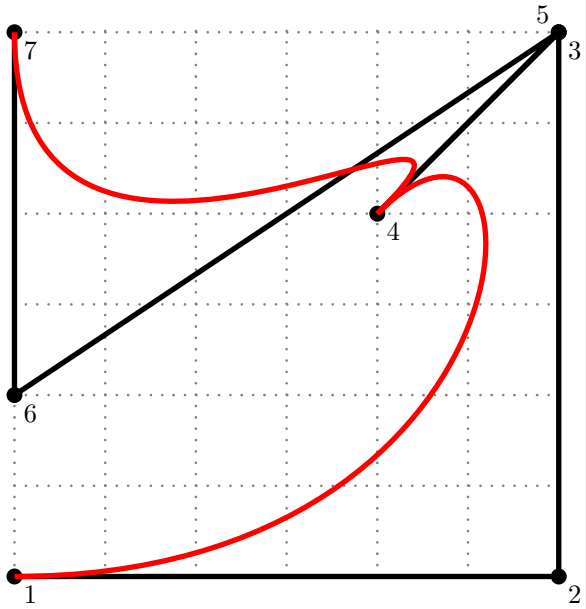
Problem 17 (4 points)

In each of the four examples below, control points are shown for two cubic Bezier curves. (Two curves share the middle point, so there are 7 points rather than 8.) Sketch out approximately what these curves will look like.



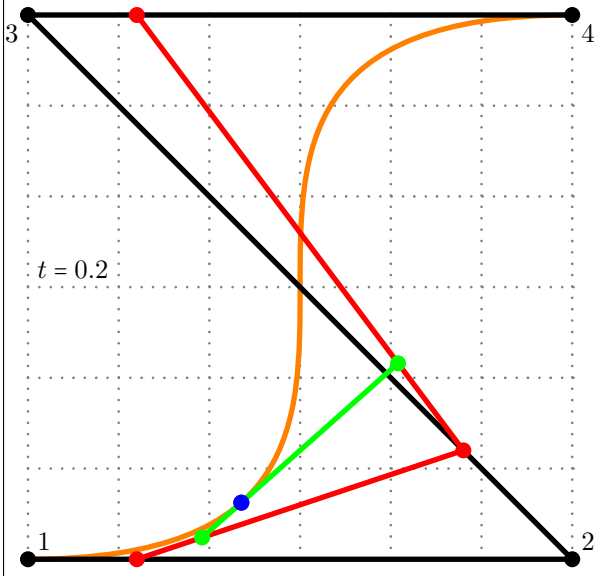


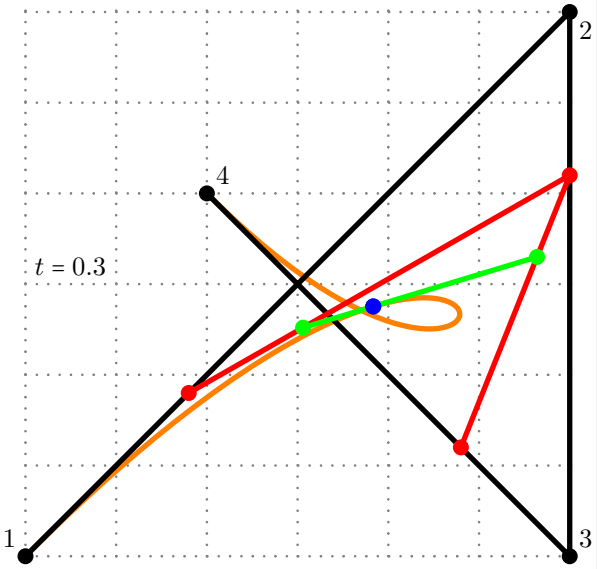




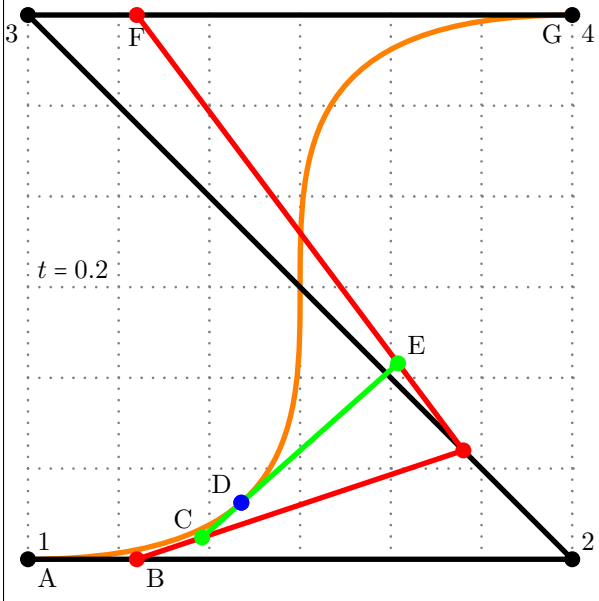
Problem 18 (4 points)

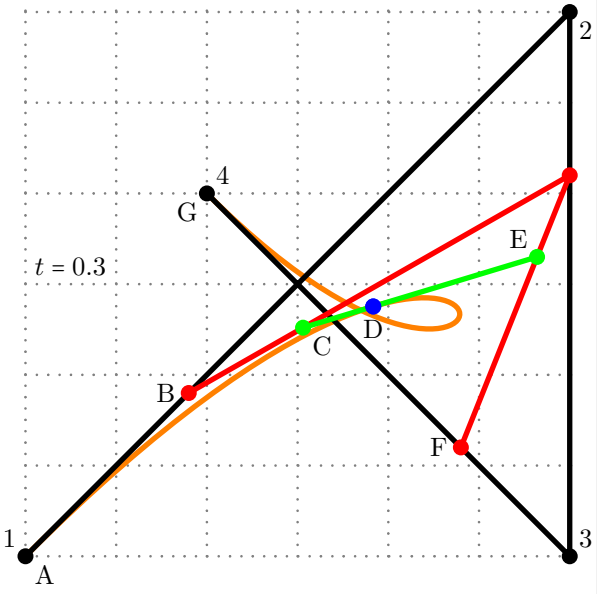
Geometrically construct the location of $P(t)$ for the Bezier curve given by the control points below.





Geometrically subdivide each Bezier curve given by the control points below and at the desired spot along the curve. Label the new control points A, B, C, ...





Problem 19 (4 points)

For each of the following discretizations, identify the differential equation that is being solved.

(a) $\dfrac{y^{n+1} - y^n}{\Delta t} = -\cos(y^{n+1})$

(b) $\dfrac{3y^{n+1} - 4y^n + y^{n-1}}{2\Delta t} = -y^{n+1}$

(c) $\dfrac{y^{n+1} - 2y^n + y^{n-1}}{\Delta t^2} = -y^n$

(d) $\dfrac{y^{n+1} - 2y^n + y^{n-1}}{\Delta t^2} + \dfrac{y^{n+1} - y^{n-1}}{\Delta t} = -y^n$

(a) $y' = -\cos(y)$
(b) $y' = -y$
(c) $y'' = -y$
(d) $y'' + 2y' = -y$