Full Name: _____

Andrew ID: _____

15-462/662, Fall 2018

Final Exam

December 14, 2018

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.
- All questions are multiple choice and have five possible answers; there is *one and only one correct answer for each question*. Points are assigned as follows:
  - Correct answer: +1 point
  - No answer: 0 points
  - Wrong answer: -1/4 point
- Good luck!

TOTAL SCORE

Question 1. Suppose a bunch of rubber balls of different colors are bouncing around the room. You don't care about *where* they bounce, but otherwise want to record complete information about their behavior, per unit time. Which radiometric quantity provides the best analogy for the quantity you want to record?

- **A.** spectral radiance
- **B.** spectral flux
- **C.** spectral flux density
- **D.** radiant flux density
- **E.** irradiance

Question 2. Taking human perception into account, which of the following is *NOT* an effective strategy for rendering a nicer-looking image in a shorter amount of time?

- **A.** add a high-resolution render of Cb and Cr color channels to an upsampled render of the Y′ channel
- **B.** increase the sampling density near the point where the viewer's gaze is focused
- **C.** use an importance sampling strategy that shoots more rays toward peaks in a surface's BRDF
- **D.** use an importance sampling strategy that shoots more rays toward objects that are greener in color
- **E.** use stratified sampling

Question 3. Which of the following color models is most appropriate for alpha compositing using the "over" operator?

- **A.** RGB
- **B.** CMYK
- **C.** SML
- **D.** HSV
- **E.** Pantone

Question 4. Suppose you have a basic halfedge data structure that just stores the `next` and `twin` pointers for each halfedge, and nothing else. Given a manifold triangle mesh of a sphere with 10000 vertices, *roughly* how many pointers (total) will you need to store?

- **A.** 20,000 pointers
- **B.** 30,000 pointers
- **C.** 60,000 pointers
- **D.** 120,000 pointers
- **E.** cannot be determined from this information

Question 5. Consider the ordinary differential equation

$$\tfrac{d}{dt}u(t) = u^2(t) + 1,$$

where u is a real-valued function of t. Suppose we want to integrate this equation using backward Euler. Letting $\tau > 0$ be the time step, and letting u_k be the value of u at the kth time step, what is the correct update rule?

A. $u_{k+1} = (1 + \sqrt{1 - 4\tau u_k - 4\tau^2})/(2\tau)$

B. $u_{k+1} = (1 - \sqrt{1 - 4\tau u_k - 4\tau^2})/(2\tau)$

C. $u_k = (1 + \sqrt{1 - 4\tau u_{k+1} - 4\tau^2})/(2\tau)$

D. $u_k = (1 - \sqrt{1 - 4\tau u_{k+1} - 4\tau^2})/(2\tau)$

E. $u_{k+1} = u_k + \tau u_k^2 + \tau$

[Note: the next four questions all refer to the scenario described here.] In class, we talked about how you can turn a triangle mesh into a mass-spring system by treating every edge as a spring. Suppose we instead treat every *face* as a "triangle spring," which penalizes the deviation of area rather than length. The potential energy for such a spring can be expressed as

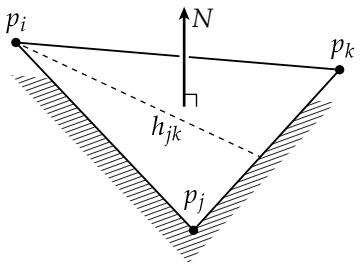
$$U := \tfrac{1}{2}(A - A_0)^2,$$

where A is the current are and A_0 is the rest area. The kinetic energy can be expressed as

$$K := \tfrac{1}{2}A\dot{c}^2,$$

where c is the barycenter of the triangle, *i.e.*, the point with barycentric coordinates $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$.

Question 6. Let $p_i, p_j, p_k \in \mathbb{R}^3$ be the current locations of the three vertices of the triangle ijk. Which of the following is *NOT* a correct expression for the current triangle area A? Here, ℓ_{ij} denotes the length of edge ij, h_{ij} is the length of the height orthogonal to edge ij, and N is the unit normal, oriented with respect to the right-hand rule; $|\cdot|$ denotes the usual Euclidean norm, and $\langle \cdot, \cdot \rangle$ is the usual Euclidean inner product.



A. $\frac{1}{2}\ell_{jk}h_{jk}$

B. $\frac{1}{2}|(p_i - p_j) \times (p_k - p_j)|$

C. $\frac{1}{2}\langle N, (p_k - p_j) \times (p_i - p_j)\rangle$

D. $\frac{1}{2}\det(p_i - p_j, N, p_k - p_j)$

E. they are all correct

Question 7. Letting $\dot{p}(x)$ denote the velocity at a point x contained inside the triangle, its kinetic energy can be expressed as

$$K := \tfrac{1}{2} \int_{ijk} |\dot{p}(x)|^2 \, dA,$$

i.e., by integrating the square of the velocity at each point over the whole triangle. This integral can be exactly evaluated using the three quadrature points $x_1 = (2/3, 1/6, 1/6)$, $x_2 = (1/6, 2/3, 1/6)$, $x_3 = (1/6, 1/6, 2/3)$ and weights $w_1 = w_2 = w_3 = 1/3$. Which of the following gives a correct expression for the total kinetic energy?

A. $\frac{1}{9} A |\dot{p}_i + \dot{p}_j + \dot{p}_k|^2$

B. $\frac{1}{18} A |\dot{p}_i + \dot{p}_j + \dot{p}_k|^2$

C. $\underline{\frac{1}{12} A \left(|\dot{p}_i|^2 + |\dot{p}_j|^2 + |\dot{p}_k|^2 + \langle \dot{p}_i, \dot{p}_j \rangle + \langle \dot{p}_j, \dot{p}_k \rangle + \langle \dot{p}_k, \dot{p}_i \rangle \right)}$

D. $\frac{1}{12} A \left(|\dot{p}_i|^2 + |\dot{p}_j|^2 + |\dot{p}_k|^2 + 2\langle \dot{p}_i, \dot{p}_j \rangle + 2\langle \dot{p}_j, \dot{p}_k \rangle + 2\langle \dot{p}_k, \dot{p}_i \rangle \right)$

E. none of the above

Question 8. Which of the following gives the correct expression for the spring force on vertex i corresponding to the potential U given above?

A. $\frac{1}{2} N \times (f_j - f_i)$

B. $\frac{1}{2} N \times (f_j - f_k)$

C. $(A - A_0) N \times (f_j - f_i)$

D. $\underline{(A - A_0) N \times (f_j - f_k)}$

E. none of the above

Question 9. Suppose we now work out the rest of the Euler-Lagrange equations, and implement a numerical integrator based on the forward Euler method. However, after running the integrator for a long time, we discover it blows up! What's a fix we can use that both avoids this issue and adds some damping to our solver?

A. scale down the kinetic energy

B. use symplectic Euler

C. <u>use backward Euler</u>

D. take smaller time steps

E. take larger time steps

Question 10. Consider a quadratic Bézier curve given by a linear combination of the bases

$$\begin{aligned} B_0(t) &:= t^2 \\ B_1(t) &:= 2t - 2t^2 \\ B_2(t) &:= 1 - 2t + t^2 \end{aligned}$$

with respect to the three coefficients $p_0, p_1, p_2 \in \mathbb{R}^2$. Which of the following pieces of code correctly evaluates the derivative of this curve at the given value of t? (You may assume that $0 \leq t \leq 1$.)

A.
```
Vector2D quadraticBezierDerivative( Vector2D p[3], double t ) {
   Vector2D u = p[1] - p[0];
   Vector2D v = p[2] - p[1];
   return (1.0-t)*u + t*v;
}
```

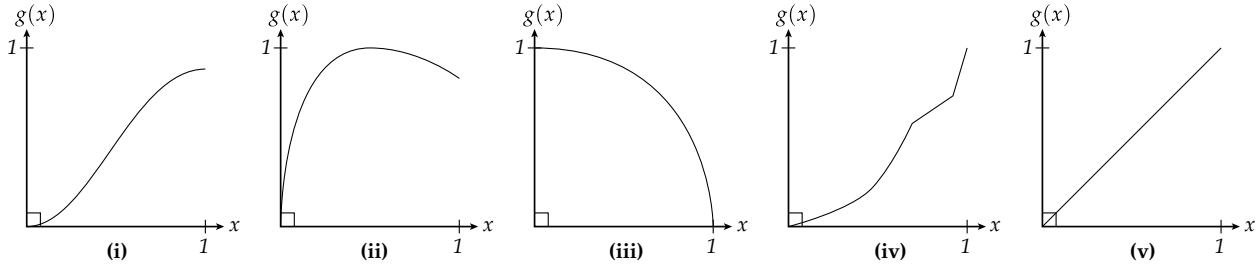
B.
```
Vector2D quadraticBezierDerivative( Vector2D p[3], double t ) {
   Vector2D u = 2.0*p[1] - p[0];
   Vector2D v = p[2] - 2.0*p[1];
   return (1.0-t)*u + t*v;
}
```

C.
```
Vector2D quadraticBezierDerivative( Vector2D p[3], double t ) {
   Vector2D a = (p[2] - p[0]);
   Vector2D b = p[0] - 2.0*p[1] + p[2];
   return 2.0*a + 2.0*t*b;
}
```

D.
```
Vector2D quadraticBezierDerivative( Vector2D p[3], double t ) {
   Vector2D a = 2.0*(p[1] - p[2]);
   Vector2D b = 2.0*p[0] - 4.0*p[1] + 2.0*p[2];
   return a + t*b;
}
```

E. None of the above.

Question 11. Which of the functions above could be a cumulative density function (CDF) for some probability density function (PDF)?



- **A.** (i) and (v)
- **B.** (i), (iii), and (v)
- **C.** (i), (iv), and (v)
- **D.** (iv) and (v)
- **E.** all but (ii)

Question 12. Suppose you want to use path tracing to render an image of a glossy, textured teapot lit via image-based environment lighting. Which of the following techniques will *NOT* help to improve the speed or quality of your render?

- **A.** importance sampling the light
- **B.** importance sampling the BRDF
- **C.** MIP mapping
- **D.** stratified sampling
- **E.** they will all help

Question 13. Imagine you've just been hired at Pixar to implement a distributed version of Scotty 3D's path tracer across a large cluster of k extremely fast machines, but where communication between machines is extremely slow. Which of the following is *NOT* a good strategy for improving the speed of distributed rendering?

- **A.** Split up the image into k pieces, have each machine render one piece, then composite the pieces into a final image.
- **B.** Have each machine render the whole image, each one using a different random seed, then take the average of these images to get the final image.
- **C.** For very large meshes, distribute each bounding volume hierarchy query across multiple machines.
- **D.** Independently re-compute the bounding volume hierarchy on every single machine.
- **E.** Have the ith machine render an image using only paths of length $nk + i$ (for $n \in \mathbb{N}$), then take the sum of these images to get the final image.

Question 14. Suppose you are building a system to simulate and render photorealistic, physically accurate clouds. Which of the following geometric representations would likely be most suitable, assuming that you only get to use one representation for both simulation and rendering?

- **A.** Bézier patches
- **B.** algebraic surfaces
- **C.** point clouds
- **D.** polygonal meshes
- **E.** regular grids

Question 15. For which of the following tasks does RGB model provide a "good enough" representation of color? (Assume that you must use RGB as the *only* color model throughout the entire task.)

- **A.** predicting the appearance of a garment of clothing under indoor lighting
- **B.** synthesizing images of virtual objects for an augmented reality headset
- **C.** rendering a preview of a multi-color 3D print
- **D.** estimating the chemical composition of Jupiter's moons
- **E.** verifying the authenticity of a 16th century painting

Question 16. Consider a manifold quad mesh without boundary, *i.e.*, a mesh made entirely of faces with four edges, where each edge is contained in exactly two faces, and the faces containing a given vertex can be given a single cylic ordering. If this mesh contains a *very large* number of quadrilaterals, what is the *approximate* ratio V:E:F of vertices to edges to faces? (You may assume that the genus is small, *e.g.*, no more than 2 or 3.)

- **A.** 1:4:2
- **B.** 1:2:4
- **C.** 1:2:1
- **D.** 1:1:2
- **E.** 1:1:4

Question 17. For a path tracer like the one you implemented for A3, which of the following scenes has the most challenging illumination?

- **A.** A mountain landscape viewed through a pinhole camera.
- **B.** A messy bedroom lit by a point light source.
- **C.** A boat lit by the head lamp of an underwater scuba diver.
- **D.** A recursively defined tree with a diffuse material, lit by a large area light source.
- **E.** The Boolean difference of two polygon meshes with a perfect specular material, lit by environment lighting.

Question 18. Suppose you're simulating a fast-moving volcanic eruption that's going to be displayed at 30 frames per second (fps). Which of the following strategies is sufficient to avoid visual artifacts, while also requiring the least computational effort? (You can assume that you have a tight upper bound on the largest temporal frequency based on an *a priori* analysis of the mechanics of volcanic eruptions.)

A. Just simulate motion at the display rate (30fps), then render.

B. Simulate all motion at 60fps, subsample it down to 30fps, then render.

C. Simulate all motion at 2x the largest temporal frequency, subsample it down to 30fps, then render.

D. Simulate all motion at 2x the largest temporal frequency *or* one over the smallest stable time step (whichever is bigger), subsample it down to 30fps, then render.

E. Simulate all motion at 2x the largest temporal frequency *or* one over the smallest stable time step (whichever is

Question 19. The code listing below tries to check whether a given triangle mesh is oriented and manifold. Which lines have errors? You may assume that there is no boundary (*i.e.*, every edge is contained in at least two triangles). You may also assume that the values stored in `triangles` are all between 0 and `nVertices`-1 (inclusive), and that every value in this range appears at least once.

```
1  #include <map> // associative array --- m[key] = value
2  #include <array> // fixed size array
3  #include <vector> // variable length vector
4
5  typedef array<int,3> Triangle;
6
7  bool isManifold( vector<Triangle> triangles, int nVertices ) {
8     vector< map<int,int> > link( nVertices );
9     for( Triangle t : triangles ) {
10       for( int n = 0; n < 3; n++ ) {
11          link[ t[n] ][ t[(n+1)%3] ] = t[ (n+2)%3 ];
12       }
13    }
14    for( auto l : link ) { // iterates over each element l of link
15       int i = l.begin()->first; // gets key of first entry of l
16       int n = 0;
17       do {
18          if( l.find(i) == l.end() ) return false; // value not defined
19          i = l[i];
20          n++;
21          if( n > nVertices ) return false;
22       } while( i != l.begin()->first );
23       if( n != l.size() ) return false;
24    }
25    return true;
26 }
```

A. there is an error in line 11

B. there is an error in the logic of the *do–while* loop

C. one of lines 18, 21, 23, and 25 has the wrong return value

D. there are other problems with the code

E. there are no errors in the code

Question 20. One technique to make image-based lighting faster[1] is to use prefiltering: if we know the scattering function ahead of time, then we can precompute, for each incoming direction, the irradiance due to a fixed environment light $\mathcal{I}(\omega)$, which is parameterized by the ray direction ω. At render time, the contribution of the environment light is just a single lookup into the prefiltered image. Letting $\mathcal{P}(\omega_i)$ denote the prefiltered irradiance for incoming direction ω_i, which of the following quantities should we estimate?

A. $\mathcal{P}(\omega_i) := \int_{H^2} f_r(\omega_o \to \omega_i)\mathcal{I}(\omega_o)\cos\theta \, d\omega_i$

B. $\mathcal{P}(\omega_i) := \mathcal{I}(\omega_i)\int_{H^2} f_r(\omega_i \to \omega_o)\cos\theta \, d\omega_i$

C. $\mathcal{P}(\omega_i) := \int_{H^2} f_r(\omega_i \to \omega_o)\mathcal{I}(\omega_o) \, d\omega_i$

D. $\mathcal{P}(\omega_i) := \mathcal{I}(\omega_o)$

E. none of the above

Question 21. Which of the following does *NOT* contribute to visual artifacts in the prefiltering/rendering strategy described in the previous question?

A. storing the image in gamma corrected RGB rather than linear RGB

B. storing results in an image with finite resolution

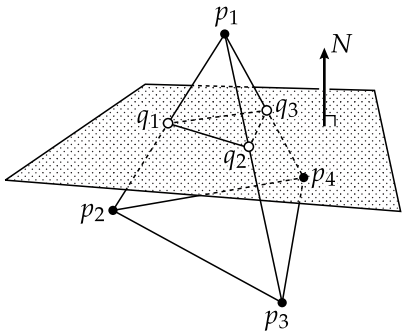
C. omission of the visibility term

D. use of Monte Carlo integration to estimate the irradiance integral

E. use of bilinear filtering to sample the prefiltered map under minification

---

Much like a triangle mesh is made up of triangles glued together along edges, a *tetrahedral mesh* is a mesh made up of tetrahedra glued together along faces, and is used to describe volumes rather than surfaces (*i.e.*, the "stuff inside" rather than just the "shell"). Since visualizing data directly on a volume is hard, a common technique is to just look at 2D slices given by the intersection of the mesh and a plane. The method below computes the intersection between a single tetrahedron with vertices (p_1, p_2, p_3, p_4) and a plane described by the implicit equation $\langle N, x\rangle - c = 0$. The output $(q_1, \ldots, p_n)$ should give the vertices of the polygon where the tetrahedron and the plane intersect (in consecutive order); orientation does not matter, and if there is no intersection then q should be empty. You should assume that all data is in "general position," *i.e.*, the intersection is not just a single point or edge, no three vertices of the tetrahedron are contained in the same line, *etc.*



[1] This technique is in fact used in real time engines like UE4 to get more realistic material/lighting at virtually no cost.

```
1  vector<Vector3D> sliceTet( const array<Vector3D,4>& p, Vector3D N, double c )
2  {
3     array<double,4> d;
4     for( int i = 0; i < 4; i++ ) d[i] = c - dot(N,p[i]);
5
6     vector<Vector3D> q;
7     for( int i = 0; i < 4; i++ )
8     for( int j = i; j < 4; j++ ) {
9        if( d[i]*d[j] < 0. ) {
10          double t = d[i]/(d[j]-d[i]);
11          q.push_back( (1.-t)*p[i] + t*p[j] );
12       }
13    }
14
15    if( q.size() == 4 &&
16        dot( cross( q[1]-q[0], q[2]-q[0] ),
17             cross( q[2]-q[0], q[3]-q[0] ) ) < 0. )
18       swap( q[2], q[3] );
19
20    return q;
21 }
```

Question 22. Consider lines 3–13 in the listing above, which are meant to compute all the points where an edge of the tetrahedron crosses the plane. Which lines have errors?

- **A.** line 4 (computes the wrong sign for the distance)
- **B.** lines 7–8 (does not not properly loop over the set of edges)
- **C.** lines 10–11: (does not properly compute the intersection point)
- **D.** A and B
- **E.** B and C

Question 23. Consider lines 15–18 in the listing above, which are meant to ensure that the output polygon is well-formed. Which of the following are true?

- **A.** these lines perform the wrong transformation
- **B.** there is an important case that is not handled
- **C.** this check isn't actually needed
- **D.** A and B
- **E.** there are no errors in this part of the code

Question 24. (0 pts) Have you enjoyed learning about computer graphics? **YES / NO**

Question 25. (0 pts) Do you feel like a total master of the material? **YES / NO**

Question 26. (0 pts) Are you excited about exploring graphics more anyway—regardless of how confident you are about the material, or what grade you get in the class?

YES / NO

Thanks for being a great class! Get some rest over the break. :-)