CS 184/284A Spring 2024 **Name:** _____

Exam 1 (In Person) **SID Number:** _____

March 18th, 2024

Time Limit: 110 minutes

- Please carefully read and sign the Honor Code described at the bottom of this page.
- Please wait to look inside the booklet until we officially start the exam.
- Put your initials on the top of every page, in case the pages become separated during scanning.
- This exam is closed book, except for one 8.5×11 page of notes (double sided), printed or handwritten.
- This exam is 110 minutes long, and has a total of 100 points. There are 22 pages (including this cover page) and 7 parts.
- Problem difficulty varies, so don't get stuck on a time-consuming problem.
- Answer each question in the space provided. Partial credit may be given on certain problems.
- You may choose to detach the last two pages which are scratch paper. We will not be grading the scratch paper.

Problem	Points	Score
1	20	
2	15	
3	20	
4	10	
5	15	
6	20	
EC	0	
Total:	100	

Honor Code and Academic Integrity Certification

The Honor Code is the commitment and work of students individually and as a community to uphold honesty and integrity. This commitment is comprehensive. Specifically on exams, it means not giving or receiving any help, or using any resources that are not permitted. The Honor Code requires that students take an active role in seeing to it that others as well as themselves uphold the letter and spirit of the Honor Code.

By signing I certify that:

- All the work submitted in my name for this exam is my work alone.
- I have not given or received any help during this exam.
- I have not used any resources during this exam aside from my own page of notes.
- I will inform course staff immediately about any Honor Code violations I become aware of.

I understand that a false statement on this declaration is a violation of the Honor Code, and would be subject to the highest level of disciplinary sanction under the University Code of Conduct.

Signature: _____

Date: _____

1. (Total : 20 points) True / False

   Mark each statement True or False. (1 point each)

   (1a) T ◯ F ◯ **[Soln: T]** We get no aliasing if we sample at more than twice the highest frequency in the signal.

   (1b) T ◯ F ◯ **[Soln: F]** In supersampled rasterization of N triangles, it is best to supersample and resolve each triangle into the framebuffer, one-by-one.

   (1c) T ◯ F ◯ **[Soln: F]** Rotating and then translating is the same as translating and then rotating.

   (1d) T ◯ F ◯ **[Soln: F]** Scaling and then rotating is the same as rotating and then scaling.

   (1e) T ◯ F ◯ **[Soln: T]** A mipmap requires $\frac{4}{3}$ the amount of memory compared to storing just the highest resolution texture image.

   (1f) T ◯ F ◯ **[Soln: F]** When starting to rasterize a new frame, we must reset values in the Z-buffer to zero.

   (1g) T ◯ F ◯ **[Soln: T]** In Catmull-Rom spline interpolation given 10 points, the resulting curve passes through all the given points, except the first and last.

   (1h) T ◯ F ◯ **[Soln: T]** If we use the de Casteljau algorithm to evaluate one point on a cubic Bezier curve, we will be given four points and calculate six interpolated points.

   (1i) T ◯ F ◯ **[Soln: T]** If we start with N triangles, after K levels of Loop subdivision, we will have $N \cdot 4^K$ triangles.

   (1j) T ◯ F ◯ **[Soln: F/T]** If we perform an Edge Collapse mesh operation, we will delete two faces.

   (This statement is only always true when working with triangle meshes, but is no longer the case with other shapes. However, because this class primarily works with triangle meshes in lecture, discussion, and projects, both answers receive credit)

   (1k) T ◯ F ◯ **[Soln: T]** Computing the intersection between a ray and a sphere involves solving a quadratic equation.

   (1l) T ◯ F ◯ **[Soln: F]** In a Bounding Volume Hierarchy, the bounding box stored at a given node is equal to the larger of the bounding boxes for its two children.

   (1m) T ◯ F ◯ **[Soln: T]** Irradiance is measured in watts per square meter.

   (1n) T ◯ F ◯ **[Soln: T]** Illuminance is measured in lux.

   (1o) T ◯ F ◯ **[Soln: F]** A hemisphere subtends 4π steradians.

   (1p) T ◯ F ◯ **[Soln: T]** We can obtain an unbiased estimate of the area under a curve from a to b by averaging random samples of the function over $[a, b]$ and multiplying by $(b - a)$.

   (1q) T ◯ F ◯ **[Soln: T]** With effective importance sampling, the variance of our randomized integral estimate goes down.

   (1r) T ◯ F ◯ **[Soln: F]** Your friends hard-code their pathtracer to terminate after 2 bounces of light, but you use Russian Roulette for an unbiased solution. Their images look brighter than yours.

   (1s) T ◯ F ◯ **[Soln: T]** A microfacet material that is very shiny, close to a mirror, has a normal vector distribution function that is concentrated around the macro-surface normal vector.

   (1t) T ◯ F ◯ **[Soln: T]** A constant-valued BRDF represents a material with a matte appearance.

2. (Total : 15 points) Graphics Pipeline

(2a) Rasterization

Your job is to write a rasterizer that draws filled circles. You are given the following data structures. Assume that the coordinate system is the same as in lecture and first homework; that is, the origin is in the top left corner of the framebuffer pixel grid, and each pixel is one unit.

```
struct Point2D {
    float       x, y;
};

struct Color {
    float       R, G, B;
};

struct Circle {
    Point2D     center;
    float       radius;
    Color       color;
};

struct Framebuffer {
    // pre-allocated memory for width * height elements
    vector<Color>   pixels;
    int             width;
    int             height;
};
```

2a.i. [2 points] Write a helper function that tests whether a given point, p, is inside a circle, c.

```
bool InsideCircle(const Point2D &p, const Circle &c) {
    // Write your code here









}
```

Solution:

```
bool InsideCircle(Point2D &p, Circle &c) {
    float dx = (p.x - c.center.x);
```

```
        float dy = (p.y - c.center.y);
        return (dx * dx + dy * dy) < (c.radius * c.radius);
    }
```

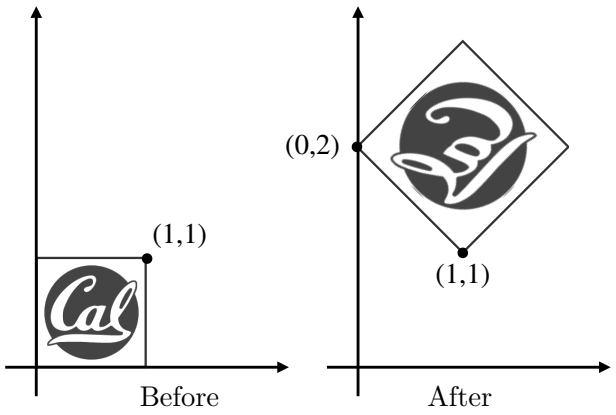
2a.ii. [5 points] Write a function that rasterizes a given, filled circle into the framebuffer. You do not need to clear the framebuffer, and you should overwrite anything in the framebuffer. You should call the helper function in the previous part of this question. You should sample fewer than all the pixels in the framebuffer.

```
// Copy of data structure definitions from
// previous page for your convenient reference.

struct Point2D {
    float       x, y;
};

struct Color {
    float       R, G, B;
};

struct Circle {
    Point2D     center;
    float       radius;
    Color       color;
};

struct Framebuffer {
    // pre-allocated memory for width * height elements
    vector<Color>   pixels;
    int             width;
    int             height;
};

void DrawFilledCircle(const Circle &c, Framebuffer &fb) {
    // Write your code here









}
```

Solution:
k

```
void DrawFilledCircle(Circle &c, Framebuffer &fb) {

    unsigned int i0 = floor(c.center.x - c.radius);
    unsigned int i1 = ceil(c.center.x + c.radius);
    unsigned int j0 = floor(c.center.y - c.radius);
    unsigned int j1 = ceil(c.center.y + c.radius);

    for (int j = j0; j < j1; ++j) {
        for (int i = i0; i < i1; ++i) {
            Point p { i+0.5f, j+0.5f };
            if (InsideCircle(p, c)) {
                fb.pixels[j * fb.width + i] = c.color;
            }
        }
    }
}
```

(2b) [8 points] Transforms



Following the conventions in lecture, which of the following 2D transforms will produce the result depicted in the image above. Assume that ReflectHorizontal and ReflectVertical are 2D transforms that reflect about the Y and X axes, respectively. Graph paper is included at the end of the exam (optional, and you may detach the graph paper page), though these questions can be answered without. Mark each option as True or False (2 points each).

2b.i. T ◯ F ◯ **[Soln:]**
Translate(1,1) · ReflectHorizontal · Scale($\sqrt{2}$) · Rotate(45°)

2b.ii. T ◯ F ◯ **[Soln:]**
Translate(1,1) · Scale($\sqrt{2}$) · ReflectHorizontal · Rotate(-45°) · Translate(-1,0)

2b.iii. T ◯ F ◯ **[Soln:]**
$$\begin{bmatrix}1&0&1\\0&1&1\\0&0&1\end{bmatrix}\begin{bmatrix}-1&0&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}\sqrt{2}&0&0\\0&\sqrt{2}&0\\0&0&1\end{bmatrix}\begin{bmatrix}\cos(-45°)&-\sin(-45°)&0\\\sin(-45°)&\cos(-45°)&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&-1\\0&1&0\\0&0&1\end{bmatrix}$$

2b.iv. T ◯ F ◯ **[Soln:]**
$$\begin{bmatrix}-1&-1&2\\-1&+1&2\\0&0&1\end{bmatrix}$$

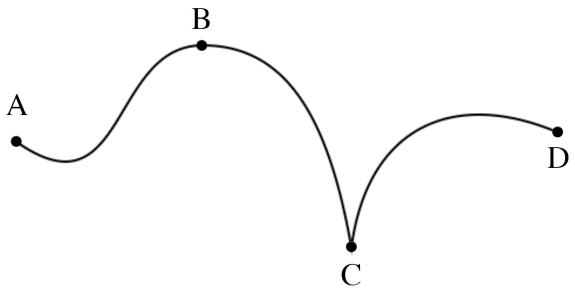
Solution: Notice that we are rotating around the bottom right corner of the original image.

- False 2b.i
- True 2b.ii
- True 2b.iii, same as 2b.ii with scale and reflection swapped – these happen to commute.
- True 2b.iv, Frame-to-World transform

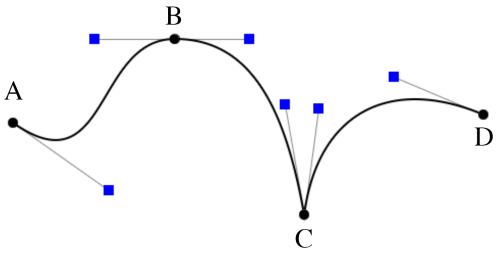
3. (Total : 20 points) Geometry

(3a) [6 points] Cubic Bezier Splines

The following diagram shows a spline composed of three cubic Bezier curves. The spline is C^1 continuous at point B. Draw the missing control points for the three cubic Bezier curves at their approximate locations.



Solution:



(3b) [4 points] Interpolation

Assume you have four data points with values 0, 2, 1 and 2. Use Catmull-Rom interpolation to derive the formula for a cubic polynomial function $f(t)$ that interpolates between the middle two values as t varies from 0 to 1.

$$f(t) = _____\ t^3 + _____\ t^2 + _____\ t + _____.$$

Solution: The derivative of the second and third points via Catmull Rom are $\frac{1}{2}(1 - 0) = \frac{1}{2}$ and $\frac{1}{2}(2 - 2) = 0$. Plugging in the point and derivatives at each endpoint into Hermite basis simplifies to

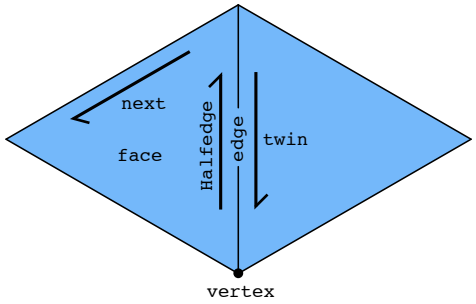
$$
\begin{aligned}
f(t) &= \begin{bmatrix} 2t^3 - 3t^2 + 1 \\ -2t^3 + 3t^2 \\ t^3 - 2t^2 + t \\ t^3 - t^2 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ \frac{1}{2} \\ 0 \end{bmatrix} \\
&= 2(2t^3 - 3t^2 + 1) + 1(-2t^3 + 3t^2) + \frac{1}{2}(t^3 - 2t^2 + t) + 0(t^3 - t^2) \\
&= 4t^3 - 6t^2 + 2 - 2t^3 + 3t^2 + \frac{1}{2}t^3 - t^2 + \frac{1}{2}t \\
&= \frac{5}{2}t^3 - 4t^2 + \frac{1}{2}t + 2
\end{aligned}
$$

The answer is $f(t) = \frac{5}{2}t^3 - 4t^2 + \frac{1}{2}t + 2$.

(3c) [4 points] Half-Edge Mesh

Assume we have a manifold mesh with consistent orientation for all triangles, stored in a Half-Edge datastructure as studied in class. Write a function that processes all the vertices surrounding (that is connected by an edge to) a given vertex. For reference, here is the slide for the Half-Edge definition from lecture slides. Tip: recall that a halfedge's vertex is the origin rather than the destination, as shown in the diagram.

```
struct Halfedge {
   Halfedge *twin,
   Halfedge *next;
   Vertex *vertex;
   Edge *edge;
   Face *face;
}
struct Vertex {
   Point pt;
   Halfedge *halfedge;
}
struct Edge {
   Halfedge *halfedge;
}
struct Face {
   Halfedge *halfedge;
}
```



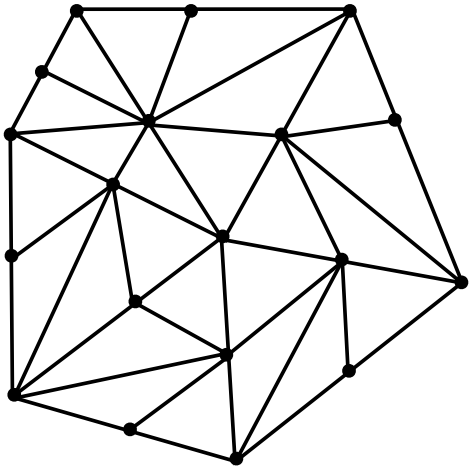

```
// Function your code should call on each surrounding vertex's Point
void process(Point &pt);

void ProcessAllSurroudingVertices(Vertex *v)
{
    // Write your code here









}
```

Solution:

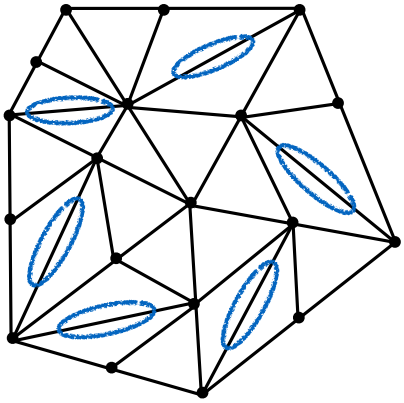
```
void ProcessAllSurroudingVertices(Vertex *v)
{
    Halfedge * h = v->halfedge;
    do {
        process(h->twin->vertex->pt);
        h = h->twin->next;
    } (while h != v->halfedge);
}
```

(3d) [6 points] Loop Subdivision

We studied a way to implement a round of Loop Subdivision on a triangle mesh. In the first step we split each edge, and in the second step we flip each new edge that connects an old point to a new point. Below you see such a mesh after the splitting step and before the flipping step. Circle all the edges that need to be flipped.



Solution:



4. (Total : 10 points) Radiometry and Photometry

(4a) [2 points] Photometric Units

Which of the following is not the same as the others?

A. Nit

B. Candela $\cdot\, m^2$

C. Lux/steradian

D. Lumen/m^2/steradian

(4b) Solid Angle and Irradiance

A piece of paper lies flat pointing straight up at a perfectly diffuse white dome that is uniformly lit.

A square appears, centered directly above a point on the sheet of paper. The square blocks light from part of the dome from reaching the paper. The square has a width of 100 cm, and is 50 cm above the point in question.

4b.i. [4 points] Calculate the solid angle, Ω, blocked by the square from the perspective of the point on the paper that is centered below the square. Hint: relate the square blocker to an imaginary cube of 100 cm width enclosing the point.

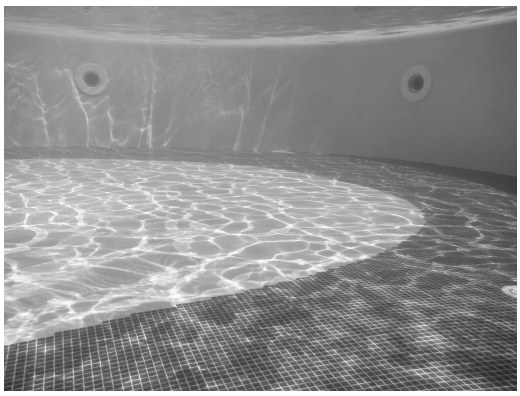
$\Omega =$ _____$\frac{2}{3}\pi$_____

Solution: The square is equal to the top face of a cube with the point at the center. The solid angle subtended by the cube is the same as a sphere, 4π. The solid angle subtended by just the top face is one sixth of this amount.

4b.ii. [4 points] Recall that a hemisphere has a solid angle of 2π steradians, so the fraction of the hemisphere blocked by the square is $\frac{\Omega}{2\pi}$. Assume the irradiance at the point in question was originally 1 W/m^2. After the square appears, is the irradiance more, less or equal to $\left(1 - \frac{\Omega}{2\pi}\right)$ W/m^2? Explain your answer briefly.

Solution: Less. $\left(1 - \frac{\Omega}{2\pi}\right)$ is the percentage of the hemisphere that remains unblocked after the square appears. If all light from the dome contributed equally to the irradiance at the point, then the irradiance would be $\left(1 - \frac{\Omega}{2\pi}\right)$ W/m^2. However, because of Lambert's cosine law, most of the irradiance at the point comes from the blocked solid angle directly overhead (small $\cos\theta$ of the angle θ between the surface normal vector and the light direction). Therefore, the irradiance will be less than $\left(1 - \frac{\Omega}{2\pi}\right)$ W/m^2. An intuitive way to think about this is that light from the top of the dome arrives directly at the point on the paper, but light from the sides of the dome arrive at a glancing angle and therefore deliver less power to the surface.

5. (Total : 15 points) Geometric Acceleration Structures for Photon Mapping



Photon mapping is a global illumination algorithm invented by Henrik Wann Jensen. It excels at simulating caustics, which are spatial patterns of concentrated light caused by optical focusing of light, such as in the images above.

In photon mapping, millions of "photons" are traced from light sources through mirror / glass objects in the scene, refracting until they intersect diffuse surfaces, where the photon is stored at the intersection point along with metadata.[1] The photon map is used later in modified path-tracing algorithms, where the irradiance on diffuse surfaces is estimated by summing up photons near the ray-surface intersection point. This produces caustic patterns where photons are dense because of optical focusing.

A crucial geometric query in photon mapping is to find all the photons within a small ball – close to a given intersection point. In this problem, you'll implement and accelerate this query function.

[1] Not important for this problem, but the metadata includes color, power and incident vector of the photon.

Here are the base structures you will use:

```
struct Point3D {
    float   x[3];
};

struct Photon {
    Point3D       pos;
    Metadata *  data;
};

struct Ball {
    Point3D   center;
    float   radius;
};
```

(5a) [4 points] First, implement the simple, exhaustive approach. Write a function that finds all the `Photons` within the given ball. Update the `results` parameter with the photons found.

```
// This is a helper function you can call in your code.
// You do NOT need to implement this.
// This function returns the distance between two points.
float distance(const Point3D &a, const Point3D &b);

void FindPhotonsInBall(const vector<Photon> &photons,
                       const Ball &ball,
                       vector<Photon> &results)
{
    /* Your code answer goes here */









}
```

Solution:

```
void FindPhotonsInBall(const vector<Photon> &photons,
                       const Ball &ball,
                       vector<Photon> &results)
{
    for(Photon p : photons) {
        if (distance(p.pos, ball.center) < ball.radius) {
            results.push_back(p);
        }
```

```
        }
    }
```

Next, accelerate the implementation using a Bounding Volume Hierarchy. For this part, assume that the photons are provided to you within a BVH data structure:

```
// An axis-aligned box
struct Box {
    Point3D min, max;
};

// Bounding Volume Hierarchy node
struct BVHNode {
    Box         boundingBox;
    bool        isLeafNode;

    // NULL if this is a leaf node
    BVHNode *   left;
    BVHNode *   right;

    // Only contains elements if this is a leaf node
    vector<Photon> photons;
};
```

(5b) [7 points] Write a function that finds all the `Photons` in a given ball, where the photons are provided in a `BVHNode`. Update the `results` parameter with the photons found. Use the declared helper function, which tests whether a `Ball` and `Box` intersect in 3D, and you may call your previous function `FindPhotonsInBall`.

```
// Helper function declaration -- you may call this helper in
// your solution if you wish.
bool intersects(const Box &box, const Ball &ball);


void FindPhotonsInBallBVH(BVHNode * photonMap,
                          const Ball &ball,
                          vector<Photon> &results)
{
    /* Your code answer goes here */
```

```
}
```

Solution:

```
void FindPhotonsInBallBVH(BVHNode * photonMap,
                          const Ball &ball,
                          vector<Photon> &results)
{
    if (NULL = photonMap) return;

    if (photonMap->isLeafNode) {
        FindPhotonsInBall(photonMap->photons, ball, results);
    } else if intersects(photonMap->boundingBox, ball) {
        FindPhotonsInBallBVH(photonMap->left, ball, results);
        FindPhotonsInBallBVH(photonMap->rtight, ball, results);
    }
}
```

(5c) [4 points] During rendering, `FindPhotonsInBallBVH` is called millions of times. Unfortunately, in testing, you find that this BVH-accelerated version of the function is actually slower than the exhaustive `FindPhotonsInBall` function. Both versions give correct results, though. You localize the problem to the construction of the BVH, which isn't shown above. Which of the following is most likely to be the problem? Explain your answer in 2-3 sentences.

- ◯ Splitting is performed around a spatial midpoint, but the code accidentally swaps the bounding boxes for the `left` and `right` nodes.
- ◯ Splitting is performed around the median element, but the code forgets to sort the photons first.
- ◯ None of the above.

Your explanation:

Solution: The answer is the second choice. We are told the results are correct, but the first choice will generate incorrect results. This is because the bounding boxes of the points are incorrect, so the BVH query will skip certain nodes when they contain points that are relevant. The second choice is correct because forgetting to sort the photons means that they will be allocated to the left and right subtree randomly – in the worst case, which seems to have happened, the left and right subtree will contain a salt-and-pepper mix of points, and have almost equally-sized bounding boxes. In this case, the BVH recursive query has to query both left and right subtrees with high probability, incurring additional traversal cost while obtaining little speedup.

6. (Total : 20 points) Cameras and Lenses

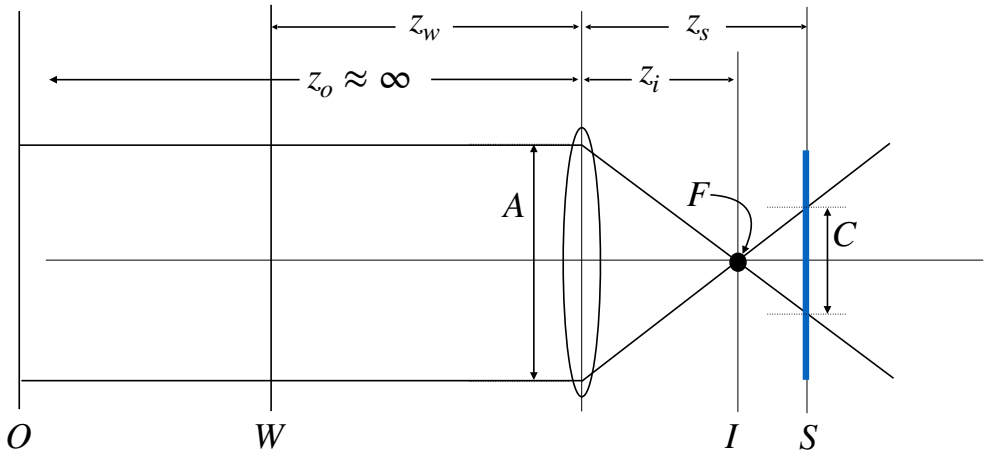
We have a camera with:

- an ideal thin lens with 36 mm focal length,
- an f/1.8 aperture,
- focused 0.3 meters away,
- pointed at some objects very far away (i.e. at optical infinity).

(6a) [10 points] Draw a cross-sectional diagram of the described situation showing and labeling at least the following items. It does not need to be accurately to scale, but should depict everything with the correct ordering and geometric relationships required to answer the next part correctly.

- A, lens aperture
- F, lens focal points
- S, sensor plane
- O, object
- W, world focal plane that that is optimally conjugate to the sensor plane
- I, image plane inside the camera that is optimally conjugate to the object,
- rays from the object passing through the lens at extremal angle, refracting into the camera and intersecting the sensor plane
- C, the diameter of the circle of confusion on the sensor plane, formed by light from the object
- distances z_s, z_o, z_w, z_i, for the separations between the lens and, respectively, the sensor plane, object plane, world focal plane and image focal plane

Solution:



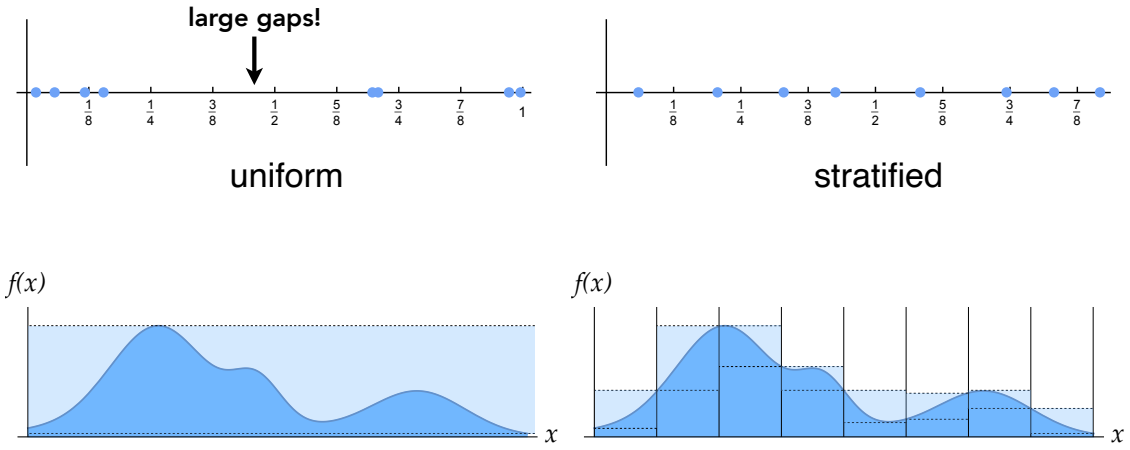
(6b) [6 points] Write down an expression for the diameter of the circle of confusion (in mm) that the distant objects make on the sensor surface. Your expression should consist of numbers and arithmetic only, so it should reduce directly to a decimal answer with a calculator. You do not need to calculate a final number.

$$C = \underline{\qquad\qquad\qquad

7. **Optional Extra Credit Challenge Questions (Maximum of 2 Bonus Points Per Part)**

Important: These problems are entirely optional, and are intended as challenges if you finish the exam early. They are worth a paltry amount of bonus points compared to the effort to answer them. **Do not attempt these problems** until you are completely satisfied with the rest of your exam. Any responses to these questions will be graded by the instructor.

(7a) (2 points (bonus)) Stratified sampling is a popular technique that can reduce variance in Monte Carlo integration. It is based on splitting the integration domain into subsets, and sampling randomly within each subset. For example, the diagram below compares drawing N samples on a line interval, using either uniform random sampling, or stratified random sampling where we split the interval into N buckets. Uniform random sampling results in large gaps between samples and clumping, which stratified sampling avoids. It is a fact that the variance of Monte Carlo integration stratified sampling is never higher than uniform random sampling (it is often lower).[2] Prove this fact for the case of drawing N samples to estimate $\int_0^1 f(x)dx$. Hint: look at the bottom right diagram, and consider how the variance of $f(x)$ within each bucket ("stratum") compares to the variance of $f(x)$ across [0,1].



Write your proof on the next page.

[2]However, note that stratified sampling reintroduces the "curse of dimensionality" that motivated us to apply Monte Carlo integration rather than quadrature in lecture.

Proof:

Solution:

(7b) (2 points (bonus)) High-Quality Pixel Resampling Filters

In lecture we briefly touched on the idea that rather than simply averaging samples within the square "boundary" of a framebuffer pixel, we could instead take a weighted average, even including samples outside the square and weighting some of them negatively. The weights to use are given by a *resampling kernel* function, $k(x, y)$. For example, a high quality kernel function is $k(x, y) = L(x)\,L(y)$, where $L(x)$ is the Lanczos function[3] with $a = 3$. With that choice, it turns out that $L(x)$ is only non-zero in the range $[-3, 3]$, and the value for the framebuffer pixel at index (i, j) is given by:

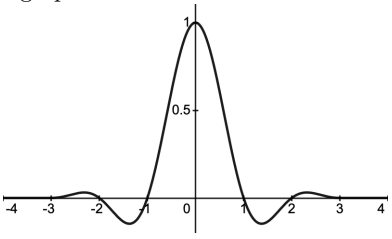
$$I_{i,j} = \int_{-3}^{+3} \int_{-3}^{+3} I(i + x, j + y)k(x, y)\,dx\,dy.$$

In your pathtracer, $I(x, y)$ is the irradiance at position (x, y) on the sensor plane. Explain how you would incorporate use of this high-quality pixel resampling filter into your pathtracing implementation. Make efficient use of all your samples.

[3]For completeness, here is the definition of the Lanczos function, with a graph for case $a = 3$:

$$L(x) = \begin{cases} \operatorname{sinc}(x)\operatorname{sinc}(x/a) & \text{if } -a < x < a, \\ 0 & \text{otherwise;} \end{cases}$$

$$= \begin{cases} 1 & \text{if } x = 0, \\ \dfrac{a \sin(\pi x) \sin(\pi x/a)}{\pi^2 x^2} & \text{if } -a \le x < a \text{ and } x \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$



This is a type of windowed sinc function. Jim Blinn recommended it (with $a = 3$) because it "keeps low frequencies and rejects high frequencies better than any (achievable) filter we've seen so far."