Problem 1: Global Illumination [12 points]
What kind of rendering techniques (radiosity or raytracing or neither) will you use in each of the following situations? Provide explanations for your answer.

1. A human hand.
   Neither, subsurface scattering is required to create realistic human hand.

2. A scene with a lots of mirrors.
   Raytracing, good for specular reflection.

3. Multiple empty glasses and bottles.
   Neither alone is sufficient, we need photon mapping to capture caustic.

4. A video moving in a house, showing its various inner structures.
   Radiosity, since its computation is view-independent and need not be recomputed.

5. Color bleeding between two walls of different color.
   Radiosity, good for color bleeding

6. A glass of milk.
   Neither, subsurface scattering in milk

Problem 2: 1D Convolution [10 points]

1. Compute $c(y) = a(x) \ast b(x)$, i.e. $c(y)$ is the convolution of $a(x)$ and $b(x)$. Simplify your answer. Show all steps. Please refer to another document for the complete answer.

Problem 3: Image Processing [12 points]

Suppose you have the following 2-D image, $X$:

\[
\begin{pmatrix}
-2 & 4 & 1 & 7 \\
-4 & 2 & 1 & 9 \\
1 & 3 & 5 & 7 \\
4 & 0 & 1 & 2
\end{pmatrix}
\]
1. Let \( k_1 \) be a kernel of the following:

\[
\begin{pmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{pmatrix}
\]

Compute \( X \ast k_1 \), i.e., the 2D convolution of \( X \) and \( k_1 \). For simplicity, you only need to compute the convolution of the non-boundary points in \( X \). In other words, the output of \( X \ast k_1 \) will only have 4 values different from the original \( X \). \[5 \text{ points}\]

Note: You should flip the kernel when doing convolution.
Note: You need to show your steps (i.e., summation of multiplications) to get full credit since it is possible to use tools such as \texttt{conv2} in Matlab to get the answer directly.

2. Explain what the kernel \( k_1 \) does when convolved with an arbitrary image. \[3 \text{ points}\]

It detects the vertical edges, just like Sobel Edge Detector. Although it has different signs compared to Sobel edge detector, it achieves the same purpose, i.e., the differences between the two sides.

3. Note that the above image has lots of noise. Will you use a gaussian filter or a median filter to clean up the noise? Explain your choice. \[4 \text{ points}\]

Median Filter is good. Note that the noise are quite random outliers, without any pattern (salt and pepper noise). Note also that most of the noises are of just one pixel. Thus, median filter can remove the noise.

Problem 4: Geometric Intersection \[16 \text{ points}\]
You learned about testing for geometric intersections in class (or you will soon). One type of intersection left out was triangle to triangle intersection.

1. Give an algorithm for testing for an intersection between 2 triangles. Pseudocode is not necessary, but your explanation should be complete. You will be graded on completeness of your algorithm and the efficiency of your algorithm. To be clear, if you are given two triangles, each with 3 points positions for the vertices, return true if there is an intersection and false if not. \[8 \text{ points}\]

The simplest solution is to do 5 raycasts along the edges of the triangles (you can leave 1 arbitrary edge out). You can take a point on an edge, \( a \), as the origin and \( b - a \) as the direction. You can then
bound the ray intersection between 0 and 1. When the triangles intersect, at least 2 edges will be intersecting a triangle (edges will not necessarily be on the same triangle). This handles the coplanar case as well.

The other popular way to solve this problem is to take the two planes of the triangles and find the line of intersection. If the planes are parallel, check if coplanar then do a simple 2D overlap test. For the lines that intersect the line of intersection, find that intersection. Take these points and check if they are within the other triangle (using barycentric coordinates).

2. Make your algorithm work for any 2 N sided polygons. For example, you have to be able to tell if there is an intersection between an octagon and a hexagon. The same requirements for your answer apply. [8 points]

For the raycast solution, you take the M + N edges and do raycasts along M + N - 1 of them. There is an algorithm in the Shirley text for ray to polygon intersection.

For the line of intersection solution, you could triangulate the polygon and do the triangle intersection test on all of them. The trick was to use a triangulation that works for concave polygons.