CSE 528 Computer Graphics (Fall 2024)

Sai Nakul 116477909

Homework Two - Non-programming Part

1. Definitions and Explanations (60 points)

(1) Perspective Projection

- **Perspective Projection:** A key approach in computer graphics, perspective projection mimics human vision by making objects appear smaller as they get farther away from the observer. This projection is necessary to create convincing 3D scenes on a 2D monitor.
- **Steps:**
  - **Camera Transformation:** Points in the 3D scene are converted into camera space using the camera's transformation matrix.
  - **Projection Matrix:** These converted points are transformed from 3D world coordinates to 2D screen coordinates by applying the perspective projection matrix.
  - **Viewport Transformation:** Finally, the actual pixel coordinates on the screen are transferred to normalized device coordinates (NDC).
- **Key Features:**
  - **Camera Model:** A virtual camera records the scene as part of the perspective projection. The viewpoint is determined by the camera's direction and position.
  - **Vanishing Points:** Depth in perspective projection is created when parallel lines converge at vanishing points.
  - **Field of View (FOV):** The FOV measures the visible area of the observable world at any particular time. A narrower FOV produces a more zoomed-in effect, while a wider FOV gives a more dramatic viewpoint.
  - **Near and Far Clipping Planes:** These planes specify the rendering distances of objects. Objects closer than the near clipping plane or farther than the far clipping plane will not be visible.
- **Applications:** Perspective projection is widely used in 3D rendering, virtual reality, and animation.

(2) Normal of a Plane

- **Definition:** The normal of a plane in computer graphics is a vector perpendicular to the plane's surface. Normals are essential in various areas of 3D graphics, such as collision detection, shading, and lighting calculations.
- **Equation of a Plane:** For a given plane defined by a point $P_0(x_0, y_0, z_0)$ and a normal vector $N(a, b, c)$, the equation of the plane can be expressed as:

  $$a(x - x_0) + b(y - y_0) + c(z - z_0) = 0$$

  Any point $P(x, y, z)$ lying on the plane satisfies this equation.

- **Properties:**
  - The normal vector indicates the orientation of the surface.
  - Normalization: The normal vector can be normalized (converted to a unit vector) by dividing by its magnitude. This ensures consistency in lighting calculations.
- **Applications:** Normals are used in lighting calculations, shading, collision detection, and texture mapping.

(3) 3D Graphics Viewing Pipeline

- **Definition:** The 3D graphics viewing pipeline is a set of procedures that converts 3D scene representations into 2D images that can be displayed on screens. Each step is essential to producing a visually meaningful image.
- **Stages of the Pipeline:**
  - **Modeling:** Creation of 3D models (such as points, lines, and polygons) or more intricate models to represent 3D objects. This step often involves defining faces, edges, vertices, and using spline or volumetric representations.
  - **Transformation:** Converts local object coordinates to world coordinates for proper placement in the scene, involving translation, rotation, and scaling.
  - **Projection:** Uses perspective projection to map 3D coordinates onto a 2D plane (the screen), simulating how objects appear smaller as they get farther from the observer.
  - **Clipping:** Removes objects or parts of objects outside the viewable region (clipping volume), simplifying the scene by focusing only on what will be displayed.
  - **Rasterization:** Converts vector graphics to pixels and determines which pixels correspond to each polygon.
  - **Fragment Processing:** Applies shading and texture mapping to enhance realism.
  - **Composition:** Combines all processed fragments, manages transparency, and performs depth testing to ensure correct object visibility before final composition.
  - **Display:** The display device renders the finished image for the audience.

(4) Mid-point Algorithm

- **Definition:** The mid-point algorithm, sometimes referred to as the midway circle algorithm, is a common technique for rasterizing circles and other curves in computer graphics. It is suitable for drawing circles on a pixel-based display due to its efficiency and reliance on the principles of symmetry and incremental computations.
- **Working Principle:** To limit the number of calculations required for each pixel, the mid-point algorithm uses incremental updates to determine the points of a circle based on its mathematical equation. By leveraging the symmetry of a circle, the algorithm significantly reduces computation.
- **Equation of a Circle:** The equation of a circle with center (h, k) and radius r is given by:

$$(x-h)^2 + (y-k)^2 = r^2$$

- **Benefits of the Mid-Point Algorithm:**
  - **Efficiency:** The algorithm calculates the next pixel using only integer arithmetic and basic decision-making, which is faster than using floating-point calculations.
  - **Simplicity:** The mid-point algorithm is straightforward to develop and understand.
  - **Symmetry Utilization:** By leveraging the symmetry of a circle, the algorithm minimizes the amount of processing needed.

(5) Oblique Parallel Projection

- **Definition:** In computer graphics, oblique parallel projection is a technique used to represent three-dimensional objects on a two-dimensional plane. Unlike orthogonal projection, which preserves angles and proportions, oblique projection allows for dimension alteration to emphasize specific features.
- **Characteristics of Oblique Parallel Projection:**
  - **Projection Lines:** In oblique projection, the projection lines are parallel to the projection plane rather than perpendicular. This can distort the appearance of the object as it is projected onto the plane.
  - **Distortion:** Depending on the angle and distance of projection, the final image may appear stretched or skewed, which allows specific features of the object to be highlighted.
  - **Viewpoint:** The viewpoint in oblique projection is typically angled with respect to the object, providing a more dynamic representation compared to strictly orthogonal views.

(6) Conic Sections

- **Definition:** Conic sections are curves created when a double cone—two identical cones positioned tip to tip—intersects with a plane. These sections, including circles, ellipses, parabolas, and hyperbolas, play essential roles in modeling, rendering, and other geometric tasks in computer graphics.
- **Applications in Computer Graphics:**
  - **Rendering:** Conic sections are frequently used in rendering techniques. Parabolas can simulate projectile motion, while ellipses can depict shadows.
  - **Collision Detection:** Conic sections represent bounding volumes in collision detection methods, providing straightforward shapes for intersection verification with other objects.
  - **Curves and Surfaces:** More intricate curves and surfaces, such as Bézier and B-spline curves, are based on conic sections, essential for simulating fluid shapes and transitions.
  - **Animation:** Parabolic trajectories in animations simulate object movement under gravity, creating realistic projectile patterns.
  - **Geometric Modeling:** Conic sections are used in CAD software to accurately model architectural features, mechanical components, and other designs requiring precise curves.

(7) Solid Models and CSG Operations

- **Definition:** Constructive Solid Geometry (CSG) and solid modeling are fundamental in computer graphics for creating and manipulating three-dimensional objects with structure and volume.
- **Differences with Surface Models:** Solid models provide a detailed physical representation of three-dimensional objects, encompassing volume, unlike surface models, which only define the outer surfaces of objects.
- **Solid Model Types:**
  - **B-rep (Boundary Representation):** Uses the solid's surface boundaries—its vertices, edges, and faces—to describe it. This model is widely used in CAD applications and is suitable for complex surfaces a
  - **CSG (Constructive Solid Geometry):** Uses Boolean operations to represent solids as combinations of primitive geometric shapes. CSG allows the creation of complex models by assembling fundamental components.
  - **Voxel Representation:** Represents solids with a grid of volumetric pixels (voxels). This representation is commonly used in medical imaging and simulations.

(8) BSP

- **Definition:** In computer graphics, the Binary Space Partitioning (BSP) technique is used to efficiently manage and organize spatial data, facilitating rendering, collision detection, and visibility determination. BSP is particularly useful in complex contexts, as it partitions space into convex sets using hyperplanes.
- **BSP Tree:** A BSP tree is a binary tree where each node represents a hyperplane dividing the space into two half-spaces. The space is recursively partitioned until primitive objects (e.g., polygons) are linked to the leaf nodes, with each child node corresponding to one of these half-spaces.
- **Structure of a BSP Tree:**
  - **Nodes:** Each internal node contains a hyperplane defined by a linear equation that divides the space into two regions: the front, where points fulfill the inequality, and the back, where points do not. Leaf nodes reference geometric primitives (such as triangles or polygons) within a specific region of space.
  - **Partitioning:** A polygon is selected to define the hyperplane, and all other polygons are categorized as either in front or behind this plane. The partitioning process continues until each leaf node contains a manageable number of primitives.

(9) Octree Representation

- **Definition:** Octree representation is a spatial partitioning technique used in computational geometry and computer graphics to efficiently manage and organize 3D data. By breaking a three-dimensional space into smaller, manageable regions, it facilitates tasks such as rendering, collision detection, and visibility determination.
- **Structure of an Octree:** An octree is a tree data structure with up to eight children per node. A 3D space, usually a cube, is recursively subdivided into eight octants, each representing a smaller cubic volume. This enables the hierarchical organization of points or objects within that volume.
- **Recursive Subdivision:** When a specified condition is met, such as exceeding the maximum number of objects per node or reaching a maximum tree depth, the root cube splits into eight smaller cubes (child nodes). This process continues, forming a hierarchy of cubes that represent increasingly detailed regions of the 3D space.
- **Leaf Nodes:** The leaf nodes of the octree hold additional information, such as point data or mesh details, or references to the objects located within that octant.
- **Object Allocation:** Objects are placed in the appropriate octants based on their spatial coordinates. If an octant exceeds the threshold (such as the maximum number of objects), it is subdivided further, and objects are reallocated accordingly.

(10) Trilinear Interpolation

- **Definition:** Trilinear interpolation is a method used in numerical analysis and computer graphics to estimate values at points within a 3D grid. It is especially valuable in applications like geographical data processing, volumetric data representation, and texture mapping.
- **Concept:** Trilinear interpolation extends linear interpolation to three dimensions. It calculates the function's value at a point (x, y, z) inside a cuboid bounded by eight points (the cube's corners).
- **Steps for Trilinear Interpolation:**
  - **Identify the Cube:** Consider a unit cube defined by eight corners.
  - **Interpolation along the x-axis:** First, interpolate values at the front and back faces of the cube along the x-axis.

  - **Interpolation along the y-axis:** Interpolate the results along the y-axis.
  - **Interpolation along the z-axis:** Finally, interpolate along the z-axis to estimate the value at (x, y, z).

- **Applications:** Trilinear interpolation is used in texture mapping, volumetric rendering, physics simulations, and image processing.

(11) Surfaces of Revolution

- **Definition:** Surfaces of revolution are a particular class of surfaces produced by rotating a two-dimensional shape (or profile) around a predetermined axis. In computer graphics, this method is frequently used to model and create complex, smooth structures like vases, bottles, and other items with rotational symmetry.
- **Generation:** A curve C in the XY-plane is rotated around an axis (usually the X-axis or Y-axis) to generate a surface of revolution. The points obtained from this rotation define the final surface.
- **Types of Surfaces of Revolution:**
  - **Cylinders:** Created by rotating a line segment around an axis. For instance, rotating a vertical line segment produces a cylindrical surface.
  - **Cones:** Formed by rotating a line that joins a circular base to the cone's peak.
  - **Ellipsoids:** Generated by rotating an ellipse around one of its axes.
  - **Paraboloids:** Produced by rotating a parabola around its axis of symmetry.

(12) Subdivision Curves

- **Definition:** Subdivision curves are a powerful tool in computer graphics for creating smooth surfaces and curves from a rough set of control points. They are essential in modeling, animation, and computer-aided geometric design (CAGD).
- **Key Features:**
  - **Control Points:** A set of initial points that define the curve's shape. These points act as "anchors" influencing the final shape.
  - **Refinement Process:** Subdivision involves iteratively refining the control points to produce a smoother curve approximation. At each iteration, a new set of points is generated for the next refinement.
  - **Limit Curve:** As the number of iterations approaches infinity, the resulting curve converges to a smooth limit curve, which the renderer can utilize.
- **Well-Known Algorithms for Subdivision Curves:**
  - Chaikin's Algorithm
  - Catmull-Clark Subdivision
  - B-Spline Subdivision

(13) Quadric Surfaces

- **Definition:** Quadric surfaces are a class of three-dimensional surfaces defined by a second-degree polynomial equation. These surfaces are important in computational geometry and computer graphics due to their mathematical properties and applications in modeling and rendering.

- **Equation of a Quadric Surface:** A general quadric surface can be defined by the equation:

$$Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J = 0$$

  where $A, B, C, D, E, F, G, H, I$, and J are constants that specify the type of quadric surface.

- **Applications:**
  - **Modeling:** Quadric surfaces are often used in 3D modeling to create continuous, smooth surfaces that represent real-world objects.
  - **Collision Detection:** They serve as bounding volumes in collision detection algorithms, due to their mathematical properties.
  - **Rendering:** The equations of quadric surfaces simplify intersection calculations, making them ideal for ray tracing.
  - **Surface Approximation:** Quadric surfaces can provide a basis for approximating more complex surfaces, streamlining the rendering process.
  - **Physics Simulations:** Their mathematical properties allow precise modeling of interactions in simulations involving light, materials, and reflections.

(14) Sweeping Objects

- **Definition:** Sweeping objects is a technique in computer graphics that moves a 2D shape (profile or cross-section) along a predefined path (trajectory or spine) to create complex 3D shapes. This technique is commonly used in modeling, animation, and design due to its flexibility and efficiency in generating surfaces and solids.

- **Features:**
  - **Profile (Cross-section):** The 2D shape swept along the path, which can be any arbitrary shape such as a circle, rectangle, or more complex polygon.
  - **Path (Trajectory):** The 3D curve along which the profile is swept. The path can be a straight line, a curve, or a more complex shape, allowing for intricate designs.
  - **Transformation:** As the profile travels along the path, it may undergo translation, rotation, and scaling, adding complexity and diversity to the final shape.

- **Types of Sweeping Techniques:**
  - **Linear Sweeping:** Moving the profile along a straight line.
  - **Curved Sweeping:** Sweeping the profile along a curved path.
  - **Revolution:** Rotating the profile around an axis.
  - **Lofting:** Sweeping the profile along multiple cross-sections.

(15) Developable Surfaces

- **Definition:** In computer graphics and geometric modeling, developable surfaces are surfaces that can be flattened into a plane without undergoing distortion. This property makes them highly useful in fields such as architecture, animation, and manufacturing.

- **Features:**
  - **Flatness:** Developable surfaces can be unfolded into a flat shape without compression or stretching. Typical examples include cones, planes, and cylinders.
  - **Curvature:** These surfaces do not curve in two directions simultaneously, as their Gaussian curvature is zero. They are flat in one direction, though they may have curvature in the perpendicular direction (e.g., cylinders).

    - **Geodesics:** A developable surface's edges or curves represent the shortest paths between points, which is essential in modeling for establishing smooth connections and transitions.

- **Types of Developable Surfaces:**
    - **Planes:** The most basic type of developable surface, flat in all directions.
    - **Cylinders:** Created by moving a straight line (generator) parallel to itself along a curve (directrix). Cylinders can be open or closed (like tubes).
    - **Cones:** Formed by joining an apex point to points on a circular base, cones can be flattened similarly to cylinders.
    - **Tori and Surfaces with Holes:** Though less common, these shapes can be regarded as developable if they can be flattened without deformation.

(16) Local Illumination Models

- **Definition:** Illumination refers to the use of light in virtual environments. Local illumination models simulate how light interacts with surfaces by considering only the light directly from light sources, ignoring global lighting effects and surface interactions.
- **Concepts in Local Illumination Models:**
    - **Light Sources:** Local models consider various types of light sources, such as spotlights, directional lights, and point lights, which affect surface interaction differently.
    - **Surface Properties:** Material characteristics influence light interaction and typically include:
        - **Diffuse Reflection:** Light scattered equally in all directions from a rough surface, often modeled using Lambertian reflection.
        - **Specular Reflection:** Light that reflects in a concentrated manner, creating highlights on smooth surfaces, often modeled with the Blinn-Phong reflection model.
        - **Ambient Reflection:** A constant amount of light that mimics indirect illumination from multiple sources in the environment.

(17) Ray Tracing

- **Definition:** Ray tracing is a rendering technique in computer graphics that simulates how light interacts with objects in a scene, creating highly realistic images. It is a core method in global illumination.
- **Ray Tracing Process:** The intensity assigned to each pixel is determined by accumulating intensity contributions from nodes in a ray-tracing tree:
    - **Surface Intensity:** Each node's surface intensity is adjusted based on distance from the "parent" surface (the next node up the tree) and added to the parent's intensity.
    - **Pixel Intensity:** The pixel's final intensity is the sum of adjusted intensities at the root node of the ray tree.
- **Procedure for Ray Generation:**
    - **Ray Generation:** The camera projects rays into the scene, typically in a straight line.
    - **Intersection Tests:** The algorithm checks if rays intersect any objects in the scene, requiring mathematical calculations to determine if a ray strikes a surface.
    - **Shading:** After detecting an intersection, the pixel's color is computed based on texture, lighting, and material properties.
    - **Shadow Rays:** Rays are cast from the intersection point to each light source to check if the point is in shadow.
    - **Global Illumination:** Sophisticated ray tracing methods simulate indirect lighting, where light reflects off objects and contributes to other objects' colors.

(18) Phong Shading

- **Definition:** Phong shading is a widely used shading model in computer graphics that realistically simulates the interaction of light with surfaces, creating a depiction of reflective and shiny materials. It achieves an eye-catching effect by combining diffuse, specular, and ambient reflection.
- **Phong Surface Rendering:** In Phong shading, normals are first interpolated across the edges, then across the polygon interiors. Illumination equations are computed for each pixel, resulting in color variation across the polygon and the generation of specular highlights. This method evaluates intensity at each pixel and often involves off-line processing.

$$n_c = (1 - \alpha)n_a + \alpha n_b$$

$$n_c = \frac{n_c}{|n_c|}$$

- **Limitations:** Phong shading is computationally expensive for real-time rendering. It is primarily used in software ray tracers, where speed is already a limiting factor.
- **Applications:**
  - **Realistic Rendering:** Widely applied in video games, simulations, and animations to create realistic materials like metals, plastics, and water.
  - **3D Modeling:** Enhances model visualization in CAD applications, improving the perception of depth and surface properties.

(19) Environment Mapping

- **Definition:** Environment mapping, also known as reflection mapping, is a technique in computer graphics that uses an image of the environment as a texture map. This method simulates reflections on highly specular surfaces, commonly applied on spherical surfaces in environmental maps.
- **Concept:** Environment mapping mimics how reflective surfaces interact with their surroundings by projecting a 2D image or set of images representing the environment onto a 3D surface. This technique is used to create reflections on shiny surfaces like glass, metal, and water.
- **Key Features:**
  - **Reflection Simulation:** Simulates how a reflective surface appears in relation to the surrounding environment, adding realism to the rendering.
  - **Texture Mapping:** Applies an environment image as a texture on the surface based on the viewer's perspective.
- **Types of Environment Mapping:**
  - Spherical Reflection Mapping
  - Cubic Reflection Mapping
  - Planar Reflection Mapping

(20) Procedural Modeling

- **Definition:** Procedural modeling in computer graphics refers to the automated creation of models and textures through mathematical operations and algorithms, rather than manual creation. This approach is ideal for quickly and reliably generating complex, variable environments, such as buildings, textures, and landscapes.

- **Guidelines:** Procedural techniques are highly powerful but should be used carefully, as physical validation is often limited. For certain objects, procedural methods are essential, particularly when enhancing models with high-frequency details. Adding noise to models can make them appear more "natural."
- **Procedural Modeling Techniques:**
  - **Noise Functions:** Functions such as Perlin noise or Simplex noise are used to create natural-looking variations and textures, frequently applied to clouds, topography, and other organic patterns.
  - **L-systems:** Used especially in plant modeling, L-systems generate complex structures by applying rewriting rules.
  - **Fractal Algorithms:** Recursive methods like the Julia set or Mandelbrot set are employed to create intricate patterns and shapes.
  - **Geometric Algorithms:** Techniques like subdivision surfaces and procedural geometry generation enable the creation of complex shapes from basic primitives.
  - **Voronoi Diagrams:** By dividing space according to distance to a set of points, Voronoi diagrams model natural phenomena like cells, topography, or crystalline formations.

(21) L-systems

- **Definition:** L-systems, also known as Aristid Lindenmayer systems, were initially created to simulate plant growth processes. They are widely used in computer graphics to generate intricate natural structures and patterns, such as fractals, trees, and flowers.
- **Components of an L-system:**
  - **Alphabet:** A collection of symbols representing different components of the system, such as branches and leaves.
  - **Axiom (Initiator):** The initial string or state of the system, usually a single symbol or series of symbols.
  - **Production Rules:** A set of rewrite rules that define how each symbol can be replaced or rewritten into one or more other symbols over multiple iterations.
- **Process:** L-systems operate using iterative rewriting:
  - **Initialization:** Start with the axiom.
  - **Iteration:** Replace each symbol in the current string using the production rules to create a new string.
  - **Repetition:** Repeat the process for a predetermined number of iterations.
- **Applications:**
  - Plant Modeling
  - Fractal Generation
  - Architectural Design
  - Animation

(22) Z-buffer

- **Definition:** The Z-buffer, or depth buffer, is the most widely used algorithm for hidden surface removal in computer graphics. It is an object-space algorithm that processes each polygon individually, making it relatively easy to implement in hardware or software.
- **Function:** The Z-buffer algorithm works by rasterizing each polygon independently and determining which parts of polygons are visible on the screen during rasterization. Visibility testing is done at the pixel level.
- **Depth Buffer (Z-buffer):** The depth buffer, or Z-buffer, is a secondary buffer with the same resolution as the frame buffer, holding depth (z-coordinate) values for each pixel position. Each cell in the Z-buffer contains the z-value (distance from the viewer) of the object at that pixel.
- **Algorithm Steps:**
  1. Initialize all depth values $depth(x, y)$ to 0 and set $refresh(x, y)$ to the background color.
  2. For each pixel:
     - Get the current depth value $depth(x, y)$.
     - Evaluate the new depth value z.
     - If $z > depth(x, y)$:
$$depth(x, y) = z$$
$$refresh(x, y) = I_s(x, y)$$

(23) Free-form Deformation (FFD)

- **Definition:** Free-form deformation (FFD) is a technique that allows for flexible deformation of models by manipulating parametric surfaces, supporting arbitrary shapes, local and global deformation, continuity, and volume preservation.
- **Process:** FFD involves placing a geometric object within a local coordinate space, building a local coordinate representation, and then deforming the local coordinate space to achieve the desired geometric transformation.
- **Basic Idea:** FFD deforms space by adjusting a lattice around the object. The deformation is defined by moving the lattice's control points, as if the object were enclosed by a deformable material (like rubber). Key components include:
  - **Local Coordinate System:** Defines the space around the object.
  - **Mapping:** Determines how deformations in the lattice affect the geometry within.
- **Advantages:**
  - Smooth deformations of arbitrary shapes.
  - Local control over the deformation.
  - Fast performance for real-time applications.
- **Applications:** FFD is widely used in the game and movie industries and is integrated into nearly every 3D modeling package.

(24) Physics-based Modeling

- **Definition:** Physics-based modeling in computer graphics involves simulating physical phenomena to create realistic animations and visual effects. This approach accurately represents how objects behave under different conditions, such as gravity, friction, and impact, by combining computational techniques with principles of physics.
- **Applications:**
  - Animation and Visual Effects
  - Simulation and Virtual Reality (VR)
- **Important Techniques:**
  - **Rigid Body Dynamics:** Simulates the motion of solid objects that do not deform. The response of these bodies to forces and collisions is governed by Newton's laws, or equations of motion.
  - **Soft Body Dynamics:** Soft bodies, unlike rigid bodies, can deform. Techniques like finite element methods (FEM) and mass-spring models represent the deformation and responsiveness of materials such as rubber, cloth, and jelly.
  - **Particle Systems:** Used to simulate effects like rain, smoke, and fire by modeling a large number of small particles, each following physical laws, allowing complex behaviors.
  - **Computational Fluid Dynamics (CFD):** Models fluid behavior, frequently used in simulations to depict interactions of liquids and gases with objects and each other.
  - **Collision Detection and Response:** Algorithms detect when objects intersect or contact each other and calculate responses to update positions and resolve collisions.

(25) Rational Bézier Spline

- **Definition:** Rational Bézier splines are an extension of standard Bézier splines, incorporating weights to allow finer control over the curve or surface's shape. This feature makes them especially useful in computer graphics and design, where precise control over geometric shapes is essential.
- **Components:** A rational Bézier spline is defined using control points along with associated weights.
- **Important Features:**
  - **Weights:** Control points with higher weights pull the curve closer, affecting the final shape.
  - **Affine Invariance:** The shape and characteristics of rational Bézier curves remain consistent under affine transformations, such as translation, rotation, and scaling.
  - **Homogeneous Coordinates:** Rational Bézier splines are often represented in homogeneous coordinates to incorporate weights.
- **Applications:**
  - **Computer Graphics and Animation:** Used for modeling intricate surfaces, paths, and shapes, enabling creative designs.
  - **CAD and CAM:** Essential for creating seamless surfaces and transitions in product design.
  - **Surface Modeling:** Used in 3D modeling for character design, automotive surfaces, and other applications where rational Bézier surfaces are defined by a grid of control points and weights.

(26) Hidden Surface Removal

- **Definition:** Hidden surface removal, also called visible surface determination, is the process of identifying visible surfaces and distinguishing them from hidden surfaces. In 3D graphics, it ensures that opaque objects block or obscure objects behind them.
- **Alternative Names:** Hidden surface removal is also known as visible surface detection methods or hidden surface elimination methods. A related problem is hidden line removal.
- **Algorithms for Hidden Surface Removal:**
  - Back-face Detection
  - Painter's Algorithm
  - Ray Casting
  - Scan-line Algorithm
  - Z-buffer Algorithm
  - Area Subdivision
- **Example:** Suppose we have a polyhedron with 3 completely visible surfaces, 4 completely hidden surfaces, and 1 partially visible/hidden surface. Hidden surfaces do not contribute to the final image in graphics synthesis.

(27) Sketch-based Interface

- **Definition:** In computer graphics, a sketch-based interface allows users to create and modify digital content or 3D objects using freehand sketches. This user-friendly approach simplifies the design process through natural drawing actions, making it accessible for users, designers, and artists with limited technical expertise.
- **Components:**
  - **Freehand Input:** Users sketch shapes or outlines on a digital canvas, which the system interprets to create corresponding 3D objects or designs.
  - **Gesture Recognition:** The interface uses algorithms to recognize and interpret user-drawn lines, shapes, and gestures, converting them into executable commands.
- **Key Features:**
  - Instant Feedback
  - Shape Recognition
  - Parametric Modeling
  - Editing Tools
- **Applications:**
  - Concept Design
  - Animation and Storyboarding
  - Education

(28) Bicubic Spline Polynomials

- **Definition:** In computer graphics, bicubic spline polynomials are a powerful mathematical tool used for surface modeling and interpolation. They provide continuous and smooth surfaces, essential for creating visually appealing images in applications like computer-aided design (CAD), 3D modeling, and animation.
- **Concept:** Bicubic splines extend the idea of cubic splines to two dimensions. They consist of a grid of control points, creating a smooth surface that interpolates these points. The surface is defined by a polynomial function of degree three in both the x and y directions.
- **Properties:**
  - **Smoothness:** Bicubic splines exhibit smooth transitions across the surface because they are C^2 continuous, meaning their first and second derivatives are continuous.
  - **Local Control:** Adjusting a control point immediately impacts the surface, allowing for easy shape manipulation and refinement.
- **Applications:**
  - Surface Modeling
  - Image Resampling
  - Animation

(29) Global Illumination Models

- **Definition:** Global illumination models simulate light interaction in a scene, including both direct and indirect lighting effects. Unlike local illumination models, which consider only direct light sources, global illumination models account for light bouncing off surfaces, contributing to the overall scene lighting.
- **Approaches:** Two main approaches for global illumination are:
  - **Ray Tracing:** Simulates light rays bouncing within the scene for realistic reflections and refractions.
  - **Radiosity:** Calculates energy transfer between surfaces, particularly effective for scenes with diffuse interreflections.
- **Characteristics:** Global illumination considers interactions between all objects, resulting in more accurate rendering at a higher computational cost, often performed offline.
- **Shadows:**
  - **Hard Shadows:** Created by distant light sources (e.g., the sun).
  - **Soft Shadows:** Caused by close light sources (e.g., light bulbs or area lights).
- **Applications:** Global illumination models are essential for creating photorealistic images in movies, visual effects, and architectural visualization.

(30) Texture Mapping

- **Definition:** Texture mapping is a technique in computer graphics for adding surface details, texture, and color to 3D models, enhancing visual complexity.
- **Methods:**
  - **Surface Detail Polygons:** Adds polygons to model details directly, increasing scene complexity and slowing down rendering.

  - **Mapping a Texture:** A more popular approach where an image (texture) is applied to a model's surface, similar to wallpapering or wrapping paper.

- **Concept:** Texture mapping uses images to fill polygons in a model, performed during rasterization (backward mapping). Curved surfaces may require extra stretching or adjustments to fit textures properly.
- **Applications:** Widely used in video games, movies, and virtual reality to enhance realism.
- **Challenges:**
  - **Texture Coordinate Specification:** Manually specifying texture coordinates can be tedious.
  - **Texture Acquisition:** Photographs can have distortions, variations in reflectance, illumination issues, and tiling problems, making texture acquisition complex.

2. (40 points, 8 points for each part)

(a) Fundamental Differences Among Linear, Affine, and Projective Transformations in 2D Graphics

In 2D graphics, linear, affine, and projective transformations differ in their ability to preserve geometric properties and introduce perspective effects. Below is a summary of each type of transformation:

- **Linear Transformations:**
  - **Definition:** A linear transformation involves operations like scaling, rotation, and reflection without any translation.
  - **Transformation Matrix:** Represented by a 2×2 matrix:

$$abcd$$

  - **Properties:**
    - Preserves the origin, i.e., the origin remains fixed.
    - Maintains the parallelism of lines, and lines remain lines.
- **Affine Transformations:**
  - **Definition:** An affine transformation extends linear transformations by including translation, allowing for repositioning of objects in space.
  - **Transformation Matrix:** Represented by a 3×3 matrix (in homogeneous coordinates):

$$abt_xcdt_y001$$

    where t_x and t_y are translations along the x and y axes.
  - **Properties:**
    - Allows translation, so the origin can be moved to another point.
    - Preserves parallelism of lines.
    - Useful for tasks like rotation, scaling, reflection, and translation in 2D graphics.
- **Projective Transformations:**
  - **Definition:** Projective transformations, also called homographies, are the most general type, allowing for perspective changes.

- **Transformation Matrix:** Represented by a 3×3 matrix:

$$abcdefgh1$$

  where g and h introduce perspective distortion.
- **Properties:**
  - Does not preserve parallelism; parallel lines can appear to converge, creating a sense of depth.
  - Can map a square to any quadrilateral, useful for simulating 3D perspective in a 2D plane.

Summary of Differences:

Transformation Type	**Preserves Parallelism**	**Allows Translation**	**Allows Perspective**
Linear	Yes	No	No
Affine	Yes	Yes	No
Projective	No	Yes	Yes

Each transformation type has increasing generality, with projective transformations being the most versatile, allowing for complex effects like perspective projection.

2(b). Differences, Properties, and Use Cases of Linear, Affine, and Projective Transformations in 2D Graphics

In 2D graphics, transformations such as linear, affine, and projective serve distinct purposes, with each transformation type allowing specific operations and producing unique effects. This section elaborates on the differences among these transformations, their properties, and when each type becomes useful in practical applications of 2D graphics.

1. Linear Transformations

Properties:

- **Origin Fixed:** Linear transformations do not translate objects, meaning that the origin (0,0) remains unchanged.
- **Linearity:** Linear transformations preserve the straightness of lines. Any line in the original space remains a line after transformation.
- **Parallelism Preserved:** Lines that are parallel before transformation remain parallel after transformation.
- **No Translation Capability:** As translation is not part of linear transformations, the object cannot shift in position.

Use Cases in 2D Graphics:

- **Scaling and Resizing:** Linear transformations can uniformly or non-uniformly scale an object, making it larger or smaller.
- **Rotation:** Linear transformations are useful for rotating objects around the origin. This is frequently applied in graphic design and image processing to reorient images.
- **Reflection:** By flipping coordinates, linear transformations can reflect an object across a line, such as the x-axis or y-axis, which is valuable in creating symmetrical patterns or mirrored images.

2. Affine Transformations

Properties:

- **Translation Capability: Affine transformations allow objects to be repositioned in the 2D plane, enabling both translation and rotation.**
- **Parallelism Preserved: Like linear transformations, affine transformations also maintain parallelism. Lines that are parallel before the transformation will remain parallel afterward.**
- **Origin Not Fixed: With translation included, the origin (0,0) can be mapped to a different location in space.**
- **Preserves Ratios and Midpoints: The transformation preserves collinearity and the relative distances between points along a line, meaning midpoints remain in the middle even after transformation.**

Use Cases in 2D Graphics:

- **Translation and Positioning: Affine transformations are essential for moving objects to desired locations within the graphics space, making them useful in graphic design and animations where object positioning is crucial.**
- **Rotation and Scaling with Position Adjustment: Unlike linear transformations, affine transformations allow rotation around a point other than the origin, useful in object manipulation tasks.**
- **Shearing: Affine transformations can skew objects along the x- or y-axis, adding perspective or depth effects. This is commonly used in simulating 3D effects in a 2D plane.**

3. Projective Transformations

Properties:

- **Perspective Effects: Projective transformations allow for the convergence of parallel lines, giving the illusion of depth and creating a 3D perspective on a 2D plane.**
- **Non-Preservation of Parallelism: Unlike affine transformations, projective transformations do not maintain the parallelism of lines. Parallel lines can appear to meet at a vanishing point.**
- **Mapping Flexibility: Allows mapping of any quadrilateral to another quadrilateral, making it highly versatile for applications that require more complex transformations.**
- **Retains Straightness of Lines: Lines remain straight after transformation, even though parallel lines may converge.**

Use Cases in 2D Graphics:

- **Perspective Projection: Essential in creating depth perception on a 2D plane, as used in maps, architectural renderings, and scenes with 3D-like effects.**
- **Image Rectification and Warping: Projective transformations are used to correct or distort images, useful in applications like image stitching, perspective correction, and transforming shapes to fit a different perspective.**
- **Simulating Camera Views: In augmented reality or virtual environments, projective transformations can simulate how objects would appear from different camera angles.**

Applications

The following table summarizes the properties and applications of each transformation type: In 2D graphics, each transformation type has distinct advantages:

- **Linear Transformations are ideal for basic scaling, rotation, and reflection.**
- **Affine Transformations add flexibility with translation, making them suitable for positioning, shearing, and complex 2D manipulations.**
- **Projective Transformations enable perspective and depth effects, essential in applications that require realistic 3D representations on a 2D plane.**

This classification and understanding of transformations are essential in computer graphics, allowing for the effective manipulation of objects, simulation of depth, and creation of complex scenes.

2(c)In practice, consider Figure 1 as an example, where we are applying 2D transformations to achieve image warping and deformation. Suppose that the transformation from Image-1 to Image-2 can be described by a function $T(x, y)$, which may represent a linear, affine, or projective transformation. Our objective is to determine how to recover the transformation $T(x, y)$ and to identify all relevant parameters associated with each transformation type. This solution should be as formal as possible, with a rigorous mathematical approach.

To recover the transformation $T(x, y)$ from Image-1 to Image-2 as depicted in Figure 1, we need to determine whether $T(x, y)$ is a linear, affine, or projective transformation. Here are the steps to recover the parameters for each type of transformation.

1. Linear Transformation

A linear transformation in 2D can be represented as:

$$x'y' = abcdxy$$

where (x, y) are the coordinates in Image-1, and (x', y') are the transformed coordinates in Image-2. The transformation matrix is defined by four parameters: a, b, c, and d.

Steps to Recover Parameters:

1. **Identify at least two pairs of corresponding points $(x_1, y_1) \to (x'_1, y'_1)$ and $(x_2, y_2) \to (x'_2, y'_2)$ between Image-1 and Image-2.**
2. **Set up the following system of linear equations based on the transformation formula:**

$$\{ x'_1 = ax_1 + by_1y'_1 = cx_1 + dy_1x'_2 = ax_2 + by_2y'_2 = cx_2 + dy_2$$

3. **Solve this system of equations to find a, b, c, and d.**

2. Affine Transformation

An affine transformation in 2D includes translation and is represented as:

$$x'y'1 = abt_xcdt_y001xy1$$

where t_x and t_y are translations along the x and y axes, respectively.

Steps to Recover Parameters:

1. **Identify at least three pairs of corresponding points $(x_i, y_i) \to (x'_i, y'_i)$ for $i = 1, 2, 3$.**
2. **Set up the following equations based on the transformation matrix:**

   $\{ x'_1 = ax_1+by_1+t_xy'_1 = cx_1+dy_1+t_yx'_2 = ax_2+by_2+t_xy'_2 = cx_2+dy_2+t_yx'_3 = ax_3+by_3+t_xy'_3 = cx_3+dy_3+t_y$
3. **Solve this system to find a, b, c, d, t_x, and t_y.**

3. Projective Transformation (Homography)

A projective transformation (or homography) in 2D is represented as:

$$x'y'1 = abcdefgh1xy1$$

where $a, b, c, d, e, f, g,$ and h are the parameters that define the transformation. The resulting coordinates (x', y') are given by:

$$x' = \frac{ax + by + c}{gx + hy + 1}, \quad y' = \frac{dx + ey + f}{gx + hy + 1}$$

Steps to Recover Parameters:

1. **Identify at least four pairs of corresponding points $(x_i, y_i) \to (x'_i, y'_i)$ for $i = 1, 2, 3, 4$.**
2. **For each pair, set up the following equations:**

   $$x'(gx + hy + 1) = ax + by + c$$

   $$y'(gx + hy + 1) = dx + ey + f$$
3. **This gives a system of eight equations (two for each point pair), which can be written in matrix form:**
4. **Solve this system to determine the values of $a, b, c, d, e, f, g,$ and h.**

Summary

To recover the transformation $T(x, y)$ from Image-1 to Image-2:

- **Linear Transformation: Use two point pairs and solve a linear system to recover $a, b, c,$ and d.**
- **Affine Transformation: Use three point pairs and solve a system of linear equations to recover $a, b, c, d, t_x,$ and t_y.**
- **Projective Transformation: Use four point pairs and solve a system of equations to recover $a, b, c, d, e, f, g,$ and h.**

Each transformation type requires a different number of point pairs, and the equations vary according to the type of transformation. Projective transformations, being the most general, allow perspective effects and require the most parameters.

2(d) Could you please extend your method and your discussions to handle their possible combination and its reconstruction as far as T(x,y) is concerned?

In the context of Figure 1, where 2D transformations are applied for image warping and deformation, we are concerned with recovering the transformation $T(x, y)$ from Image-1 to Image-2. Suppose that $T(x, y)$ may involve a combination of linear, affine, and projective transformations. Here, we extend our method to handle such possible combinations and discuss how to reconstruct the transformation.

Combined Transformations: Linear, Affine, and Projective

In 2D transformations, a transformation function $T(x, y)$ may involve multiple types of transformations, either applied sequentially or in combination. The overall transformation matrix depends on the specific combination of linear, affine, and projective transformations. We will examine how each transformation type is represented, how they can be combined, and how to reconstruct the parameters for the composite transformation.

1. Linear Transformation

A linear transformation can be represented as:

$$x'y' = abcdxy$$

**where the transformation matrix ab
cd includes parameters a, b, c, and d.**

Properties: Linear transformations include scaling, rotation, and reflection but do not include translation or perspective effects. Linear transformations are useful for rigid transformations where the origin remains fixed.

2. Affine Transformation

An affine transformation extends a linear transformation by incorporating translation, and is represented as:

$$x'y'1 = abt_xcdt_y001xy1$$

where t_x and t_y introduce translation in the x- and y-directions, respectively.

Properties: Affine transformations include scaling, rotation, reflection, shearing, and translation. They preserve parallelism but not the fixed point at the origin. Affine transformations are useful for more general transformations where translation is required.

3. Projective Transformation

A projective transformation, also known as a homography, is the most general transformation and can introduce perspective effects. It is represented as:

$$x'y'1 = abcdefgh1xy1$$

where the parameters g and h introduce perspective distortion.

Properties: Projective transformations allow for perspective projection, where parallel lines can converge, creating depth perception. This type of transformation is essential for simulating a 3D perspective on a 2D plane.

Combining Transformations

When combining transformations, the overall transformation matrix T is the product of the individual transformation matrices for each type. Let:

$$T = T_{projective} \cdot T_{affine} \cdot T_{linear}$$

where:

- $T_{linear} = ab$
  cd **represents the linear transformation,**

- $T_{affine} = abt_x$
  cdt_y
  001 **represents the affine transformation, and**

- $T_{projective} = abc$
  def
  $gh1$ **represents the projective transformation.**

The composition of these transformations allows us to perform complex transformations, including translation, scaling, rotation, shearing, and perspective projection.

Reconstruction of $T(x, y)$

To reconstruct $T(x, y)$, we need to solve for the parameters of each transformation matrix. Depending on the transformation type (or combination of types), the number of parameters and the number of point correspondences required will vary.

Steps for Reconstruction:

1. **Identify Point Correspondences: Gather at least four pairs of corresponding points $(x_i, y_i) \to (x'_i, y'_i)$ between Image-1 and Image-2.**
2. **Set Up Equations: Based on the transformation matrix (linear, affine, projective, or combined), set up a system of equations to represent each point correspondence.**
3. **Solve the System: Use linear algebra techniques to solve for the parameters. For projective transformations, the system may be nonlinear, requiring specialized methods such as least squares estimation or singular value decomposition (SVD).**

Example: Combined Affine and Projective Transformation **Suppose $T(x, y)$ is a combination of affine and projective transformations. We represent T as:**

$$x'y'1 = abt_xcdt_ygh1xy1$$

For each point correspondence $(x_i, y_i) \to (x'_i, y'_i)$, we have:

$$x'_i = \frac{ax_i + by_i + t_x}{gx_i + hy_i + 1}$$

$$y'_i = \frac{cx_i + dy_i + t_y}{gx_i + hy_i + 1}$$

These equations can be used to set up a system and solve for the parameters a, b, c, d, t_x, t_y, g, and h.

Conclusion

By using the appropriate number of point correspondences and solving the resulting system of equations, we can recover the parameters of $T(x, y)$ for any combination of linear, affine, and projective transformations. This method allows us to accurately reconstruct the transformation matrix T and apply it to achieve the desired image warping and deformation.

2(e) What if T(x,y) involves some kinds of non-linear transformation, what are your thoughts on handling and re covering non-linear transformation? For this part only, you do NOT need to provide any proof, a clear description and justification of ideas would suffice for this part! 1 Figure 1: Illustration of a 2D image transformation (from Image-1 to Image-2). Certain types of transformation and/or their possible combination occur from Image-1 to Image-2. We are interested in recovering the right transformation T(x,y).

When $T(x, y)$ involves non-linear transformations, the process of recovery becomes more complex compared to linear, affine, or projective transformations. Non-linear transformations do not have a straightforward matrix representation, as they may include warping, non-uniform scaling, radial distortions, or other complex deformations that cannot be captured by linear combinations of coordinates. Here are some thoughts and approaches on handling and recovering non-linear transformations.

1. Use of Polynomial or Higher-Order Basis Functions **One approach to model non-linear transformations is to approximate $T(x, y)$ using polynomial or higher-order basis functions. For example, we can represent $T(x, y)$ as a polynomial function:**

$$T(x, y) = \sum_{i,j} c_{ij} x^i y^j$$

where c_{ij} are coefficients that need to be determined. This approach allows us to capture non-linear relationships between the input and output coordinates. By using higher-degree polynomials, we can approximate more complex distortions. However, this method requires a larger number of corresponding points and can become computationally expensive.

2. Radial Basis Functions (RBF) and Thin-Plate Splines (TPS) **Radial basis functions (RBF) and thin-plate splines (TPS) are popular techniques for handling non-linear transformations, especially for image warping and deformation. These functions allow for smooth interpolation across points while minimizing distortion, making them well-suited for cases where the transformation is non-linear.**

- **Radial Basis Functions (RBF): RBFs involve functions that depend on the distance from a central point, allowing for flexible warping based on control points. By selecting key points in Image-1 and their corresponding points in Image-2, we can use RBF interpolation to find the non-linear transformation that best matches the mapping.**
- **Thin-Plate Splines (TPS): TPS is another effective approach for non-linear transformations, especially when there is a need to minimize bending energy across control points. TPS provides a smooth, continuous transformation that interpolates given point correspondences with minimal distortion, making it ideal for complex warping and deformations.**

3. Machine Learning Models (Neural Networks) In recent years, neural networks and other machine learning models have shown success in learning complex, non-linear transformations from data. By training a neural network with pairs of corresponding points from Image-1 and Image-2, the model can learn to approximate $T(x, y)$ as a non-linear function. This approach is highly flexible and can capture intricate transformations, but it requires a sufficient amount of data (point correspondences) and computational resources for training.

4. Piecewise Linear or Affine Approximation For certain non-linear transformations, an approximation approach using a combination of piecewise linear or affine transformations may be effective. This method involves dividing the image into smaller regions and applying a simpler (linear or affine) transformation within each region. While this approach does not fully capture the global non-linear behavior, it can provide a good approximation if the non-linearity is moderate and varies smoothly across the image.

5. Optimization-Based Approach If a specific form of the non-linear transformation is known or can be approximated, we can set up an optimization problem to minimize the difference between the transformed points and the target points. For example, we can define an objective function that measures the error between $T(x, y)$ and the corresponding points in Image-2, and use gradient-based optimization techniques to iteratively adjust the parameters of $T(x, y)$ until the error is minimized. This approach is flexible and can be tailored to various types of non-linear transformations, but it may be computationally intensive and require a good initial guess for the parameters.

Conclusion

In summary, handling and recovering non-linear transformations requires more sophisticated approaches than those used for linear, affine, or projective transformations. Techniques such as polynomial basis functions, radial basis functions, thin-plate splines, neural networks, piecewise approximation, and optimization-based methods provide various strategies for approximating and recovering $T(x, y)$ when it is non-linear. The choice of method depends on the nature of the non-linearity, the available data, and computational resources.