CS 3451: Computer Graphics Notes (Midterm Version)

Raster Images

- Computer screens are composed of PIXELS arranged in rows known as SCANLINES, each with an assigned color in RGB values (0 to 255).
- Pixels are organized in a rectangular grid according to a coordinate system.

2. Coordinate System in Processing

- In Processing, coordinates are given as 2D (x, y) with X on the horizontal axis and Y on the vertical axis.
- The origin is in the UPPER-LEFT corner with X as the rightward distance and Y as the downward distance.

3. Specifying Grid Size in Processing

- The grid size in Processing is specified using the "size" command, e.g., size(400, 400) for a 400x400 pixel window.

4. Essential Processing Functions

- Every Processing program must have two functions:
    - void setup() {...}: handles initialization.
    - void draw() {...}: repeatedly called to redraw the window every frame.

5. Important Processing Commands

- rect(x, y, width, height): draws a rectangle with specified dimensions and upper-left corner coordinates.
- ellipse(x, y, width, height): creates an ellipse.
- background(r, g, b): fills the window with the given background color.
- stroke(r, g, b): sets the outline color.
- fill(r, g, b): sets the fill color.
- noStroke(): turns off outlines.
- noFill(): turns off the fill color.

6. Predefined Variables in Processing

- "width" and "height" are predefined variables referring to the width and height (in pixels) of the window.
- "mouseX" and "mouseY" are predefined variables for the mouse position coordinates.

7. Adapting Coordinate Systems in Processing

- To change the coordinate system, mapping between the desired range and Processing's coordinate system is necessary using mathematical transformations.

8. Using Trigonometry in Processing

- Processing uses radians for sine and cosine functions.
- The unit circle's X-coordinate is cos(theta), and the Y-coordinate is sin(theta).

9. Final Thoughts

- Understanding Processing and basic math concepts is essential for completing introductory projects.

Basic Linear Algebra Review and 2D Transformations

1. Introduction to Basic Linear Algebra Review

- A refresher on fundamental linear algebra concepts, emphasizing matrix multiplication, dot product, and vector operations.

2. Matrix Multiplication

- Matrix multiplication is not commutative in linear algebra; $AB \neq BA$ in general.
- The process involves multiplying respective rows from the first matrix with columns from the second.

3. Dot Product and Vectors

- The dot product of two vectors is a fundamental operation used extensively in computer graphics.
- A dot product of 0 indicates orthogonality between vectors.

4. Common Transformations

- Three fundamental transformations in computer graphics:
  - Translation: Changing an object's position without altering its shape.
  - Scaling: Altering the size of objects, potentially with respect to a specified origin.
  - Rotation: Rotating points around the origin by an angle.

5. Representation in Matrices and Vectors

- Translation, scaling, and rotation can be represented using matrices and vectors.

- Matrices for transformations allow for combining operations and achieving the desired effect on an object.

6. Homogeneous Coordinates

- Introducing homogeneous coordinates to allow for consistent matrix operations across all transformations.
- Homogeneous coordinates help handle translation as matrix multiplication, maintaining consistency with other transformations.

7. Common Transformation Matrices

- The key transformation matrices:
  - Scaling matrix (S) to alter object size.
  - Rotation matrix (R) to rotate points around the origin.
  - Translation matrix (T) to shift points.

8. Additional Transformations

- Shear: Shifting object coordinates in one direction, a less common transformation.
- Reflection: A special case of scaling involving flipping points about a line, often achieved through scaling by negative values.

9. Practical Application

- Emphasizing the importance of understanding and utilizing these matrices and operations in computer graphics to manipulate and transform objects effectively.

Transformations and Matrix Operations

1. Overview of Transformations

- Three major types of transformations discussed: translation, scaling, and rotation.
- Translation involves adding vectors or using matrix multiplication with homogeneous coordinates.
- Transformation order matters to achieve the desired effect.

2. Rotating Around an Arbitrary Point

- Explained the need to translate an object to the origin, perform rotation, and then translate it back for in-place rotation around a specified point.
- Demonstrated the matrix operations involved in this process.

3. Matrix Multiplication and Transformation Combinations

- Emphasized the right-to-left nature of matrix multiplication and the importance of understanding the order of operations.
- Combined multiple transformations into a single matrix through matrix multiplication.
- Explained how combining operations into one matrix reduces computational load.

4. Application to Polygon Transformation

- Demonstrated how to apply combined translation and scaling operations to transform a polygon using matrix multiplication.

5. OpenGL API Usage

- Provided an example of how to use the OpenGL API for transformations, emphasizing the need to follow the order of operations.

6. Commutativity of Operations

- Explained which operations commute with each other and which do not in 2D and 3D spaces.
- Highlighted the non-commutative nature of rotations in 3D and the commutativity of uniform scaling.

OpenGL and Matrix Stack

1. Introduction to OpenGL

- OpenGL is a graphics library widely used for drawing and rendering in applications like Processing.
- Basic primitives, such as lines and circles, are drawn using OpenGL commands.

2. Drawing Shapes with OpenGL Commands

- Demonstrated the usage of OpenGL commands (glBeginShape, glVertex, glEnd) to draw simple shapes like lines and circles.
- Discussed how to draw connected lines and a unit circle using these commands.

3. Matrix Stack and Transformations

- Explained the concept of the matrix stack in OpenGL, allowing for transformation operations.
- Components of the matrix stack include the Current Transformation Matrix (CTM), glPushMatrix, glPopMatrix, and transformation commands like glTranslate, glScale, and glRotate.

- Illustrated how the matrix stack facilitates transformations and coordinate system changes.

4. Usage and Purpose of the Matrix Stack

- Discussed three major uses of the matrix stack: changing coordinate systems, instantiation (object re-use), and hierarchy creation.
- Detailed how the matrix stack aids in organizing transformations, enabling complex object rendering.

5. Hierarchical Drawing with the Matrix Stack

- Illustrated an example of drawing a stick figure using the matrix stack to handle hierarchical rendering of components.
- Demonstrated how transformations are applied to various components, allowing for a hierarchical structure.

Project 1 and 3D Introduction

1. Introduction to Project 1

- Overview of Project 1, which involves creating a digital representation of a person using transformations.
- Highlighted the usage of the matrix stack to handle persistent transformations during drawing.

2. Hierarchical Drawing of the Arm

- Expanded on the drawing of a person by detailing the drawArm() and armExtent() methods.
- Demonstrated the hierarchical approach to drawing components by translating and scaling the arm.

3. Drawing the Person - Torso and Arms

- Introduced the person() method to draw the torso and arms of the person.
- Explained the usage of matrix transformations (scale, translate) to position and draw different body parts.

4. Understanding Matrix Stack in Hierarchical Drawing

- Emphasized the significance of the matrix stack in simplifying hierarchical drawing.
- Described the matrix push and pop operations to manage the transformation state for different components.

5. Transition to 3D with Right-Handed Coordinate System

- Introducing 3D space using a right-handed coordinate system with x, y, and z axes.
- Explained the homogenization of 3D vectors for transformation purposes, adding a 1 to make them 4D.

6. Rotation Matrices for 3D

- Presented the rotation matrices for 3D rotations around the x, y, and z axes.
- Showed the matrices for rotation around each axis, clarifying the column swaps in the y-axis rotation matrix.

7. 3D Projection

- Discussed the challenge of displaying 3D objects on 2D screens and the need for projections.
- Differentiated between parallel (orthographic) projection and perspective projection, explaining their characteristics.

Projections and Perspective

1. Introduction to Projections

- Definition of Projection: Moving points onto a subspace (e.g., projecting a 2D line onto the 1D X-axis in geometry).
- Two main types of projections:
  - Parallel Projection: Lines from objects are parallel when projected.
  - Perspective Projection: Projector lines converge to a point, mimicking how our eyes perceive.

2. Parallel Projection

- Imagery of a view plane perpendicular to the Z-axis, acting like a window into the 3D space.
- Projection onto the View Plane:
  - Points P(x, y, z) are projected to P(x, y, 0) on the view plane along the Z-axis.
- Viewing Transformation:
  - Maps points from view plane to screen window.
  - Mapping from a certain range on the view plane to the screen window using linear transformation.

3. Perspective Projection

- Involves triangles, angles, and similar triangles to determine projection.
- Center of Projection (COP): Virtual "eye" at a point in the grid, usually (0, 0, -1).
- Projection of Points onto View Plane:

        - Similar triangles between COP, point, and view plane yield the projected coordinates.
        - $x' = x/|z|$ and $y' = y/|z|$, indicating that objects farther away seem smaller.
- Field of View (FOV):
    - Determines the extent of what is visible on the view plane.
    - Calculating the maximum y-coordinate on the view plane for a given FOV.
    - Mapping this FOV to the screen window.

Output Devices and Liquid Crystal Displays

1. Camera Transformations for Viewing

- Camera Function in Processing:
    - Takes 9 parameters: 3 for camera position, 3 for the point being looked at, and 3 for the "up" vector.
    - Corresponds to gluLookAt in OpenGL.
- Mathematical Approach:
    - Involves translation and rotation to position and orient the camera.
    - Utilizes a translation matrix (T) to move the camera to a specified position (x, y, z).
    - Rotation matrix (R) aligns the camera's gaze direction with the negative Z-axis.
    - The final transformation matrix is given by $M_{view} = R \times T$

2. Perspective vs. Parallel Projection

- Perspective Projection:
    - Typically utilized, precise view specification.
    - Camera parameters precisely determine the viewpoint and direction.
- Parallel Projection:
    - Provides a viewing box instead of a specific point.
    - Less constrained, often used for drafting and alignment purposes.

3. Output Devices and Frame Buffers

- Output Devices and CPU Interaction:
    - CPU output sent to frame buffer in the memory.
    - Frame buffer image sent to the video controller which in turn transmits it to the monitor.
- Double Buffering:
    - Maintains two buffers to avoid tearing and ensure smooth image transition.

4. Liquid Crystal Displays (LCDs)

- Liquid Crystals:

        - Molecules that change conformation based on external factors like voltage or temperature.
- Working Principle:
    - LCDs use liquid crystals that twist in response to voltage, blocking or allowing light.
    - Polarizing filters ensure light passes through only when aligned with the crystals.
- Special Glasses:
    - Utilize polarized lenses that alternate to allow vision through specific lenses, used in 3D films with older technology.

Display Technologies and Line Equations

1. LCD Displays

- Twisted Nematic Liquid Crystals:
    - Fundamental component of LCD displays.
    - Control the passage of light by twisting in response to voltage, regulating brightness.
    - Sub-pixels for red, green, and blue are controlled independently to produce various colors.

2. E-Ink (Electronic Ink) Displays

- Working Principle:
    - Reflects light like traditional paper and canvas, containing microcapsules with charged particles (white and black).
    - Application of electric field selectively moves particles, creating a visible pattern.
- Advantages and Disadvantages:
    - Pros include high visibility in sunlight and low power consumption.
    - Cons are slower refresh rates and limited color capabilities compared to LCDs.

3. Future Display Technologies

- Emerging Technologies:
    - "Mirror display technology" uses mirrors in each pixel, common in projectors and VR devices.
    - Ongoing research in holography and digital fluorescent crystal techniques for true 3D holographic displays.
    - Experimental approaches include direct retinal projection and direct stimulation of the visual cortex.
- Sci-Fi Concepts:
    - Far-fetched concepts involve direct integration with the visual cortex, bypassing the eyes for display.

4. Line Equations

- Parametric Equation:
    - Defines a line using a parameter t that slides along the line between two points (P1 and P2).
    - Equations for X and Y coordinates in terms of t allow representation of any point on the line.
- Implicit Equation:
    - Uses a function f(x,y) to define points on a line.
    - Equation $f(x,y)=a*x+b*y+c$ for a line, where a, b, c are coefficients.
    - Equates to the familiar $y=m*x+b$ equation.

Line Drawing and Polygon Rasterization

1. Line Drawing Algorithm

- Parametric Line Drawing:
  Parametric equation used: P(t) = P0 + t * (P1 - P0).
  Parameter t is replaced by i in the code.
- Algorithm Steps:
  Calculate Delta x and Delta y between the points.
  Calculate the length of the line (length = max(|Delta x|, |Delta y|)).
  Determine increments in x and y (x_inc = Delta x / length, y_inc = Delta y / length).
  Loop from 0 to length, incrementing x and y accordingly, rounding to nearest pixel.
  Draw pixels at the calculated coordinates.

2. Line Drawing Code Example

```
void line(int x0, int y0, int x1, int y1) {
   dx = x1 - x0;
   dy = y1 - y0;
   length = max(abs(dx), abs(dy));
   x_inc = (float) dx / length;
   y_inc = (float) dy / length;

   x = x0;
   y = y0;

   for (i = 0; i < length; i++) {
      gtWritePixel(round(x), round(y), someColor);
      x += x_inc;
      y += y_inc;
   }
}
```

3. Future Directions in Class

- Rendering Progression:
    - Orthographic views -> Perspective projection -> Hidden surfaces -> Surface coloring -> Shading -> Multiple light sources -> Shadows and reflections -> Textures and images.

4. Polygon Rasterization
    - Rectangle Filling:
        - Fill a rectangle with solid color by iterating over pixels within the specified bounds and setting their colors.
    - Polygon Rasterization Approach:
        - Rasterize by filling one scanline (row of pixels) at a time from bottom to top.
        - Intersections between polygon edges and pixel grid help determine where to fill.
        - Sort the intersections on x-values and fill between them from left to right.
5. Intersection Calculation for Polygon Rasterization
    - Approach:
        - Determine the leftmost intersection point.
        - Use neighboring polygon points to create a right triangle and calculate the step needed to move along the polygon's edge for each scanline.

Hidden Surfaces and Z-Buffering

1. Hidden Surfaces and Visibility
    - Objective:
        - Determine which surfaces are visible and which are hidden in a 3D scene.
        - Essential for realistic rendering and creating accurate 3D visuals.
2. Painter's Algorithm
    - Approach:
        - Sort polygons based on their depth (Z-coordinate) or centroid's Z-value.
        - Draw polygons in back-to-front order.
    - Limitations:
        - Inaccurate when polygons intersect or have complex spatial relationships.
        - Not widely used due to these limitations.
3. Z-Buffering
    - Approach:
        - Maintain a Z-buffer, storing the closest Z-value for each pixel.
        - Compare Z-values of polygons during rasterization to determine visibility.
        - Draw the closest visible polygon for each pixel.
    - Advantages:
        - Accurate and handles complex scenes well.
        - Widespread use today due to modern memory capabilities.
4. Z-Buffering Pseudocode

```
# Setup: Initialize background color and initial Z-values for all pixels.
for every pixel (x, y):
    writePixel(x, y, background_color)
    writeZ(x, y, very_far_away_large_constant_value)
```

```
# Main Loop: Iterate over each polygon and draw the visible pixels.
for every polygon:
    Determine pixels covered by the polygon using rasterization techniques.
    for every pixel (x, y) in polygon:
        pz = Z-value of polygon at pixel (x, y)
        if pz >= ReadZ(x, y):
            # Pixel is the new closest pixel, update Z-value and color.
            writeZ(x, y, pz)
            writePixel(x, y, poly_color)
```

Surface Shading Techniques

1. Dot Product for Unit Vectors
    - Dot Product of Unit Vectors:
        - u * v = v * u = cos(theta)
        - Useful for calculating the angle between two vectors.
2. Surface Normal Calculation for a Polygon
    - Surface Normal Calculation:
        - For a triangle ABC:
            - vector1 = B - A
            - vector2 = C - A
            - Normal = crossProduct(vector1, vector2) = v1 X v2
    - Surface normal is a vector perpendicular to the surface.
3. Surface Shading Basics
    - Factors affecting Surface Brightness:
        - Light source positions and properties.
        - Surface reflectivity and absorbance.
    - Diffuse Surfaces:
        - Reflect light equally in all directions.
        - Examples: chalk, paper.
    - Light Interaction and Surface Brightness:
        - Surface area illuminated changes with angle.
        - Brightness decreases as angle between light and surface increases.
4. Calculating Surface Color
    - Formula for Surface Color:
        - final surface color = Cs * Cl * max(0, N*L)
        - Cs: Surface color, Cl: Light color, N*L: Light intensity on the surface.
5. Ambient Light and Indirect Illumination
    - Ambient Light:
        - Constant light added to all objects, simulating indirect illumination.
    - Enhanced Illumination Model:
        - C = Cs * (Ca + Cl * max(0, N*L))

- Ca: Ambient light color.

6. Specular Reflection - Phong Illumination Model
   - Specular Surfaces:
     - Reflect light in a focused direction.
     - e.g., plastics, metals.
   - Phong Illumination:
     - C = Cl * max(0, (E*R)^P)
     - E: Eye direction, R: Reflected light direction, P: Specular power.

Specular Reflection and Shading Models

1. Specular Surfaces and Phong Illumination Model
   - Diffuse vs. Specular Surfaces:
     - Diffuse surfaces reflect light equally in all directions.
     - Specular surfaces have bright spots (highlights) when reflected light is observed head-on.
   - Phong Illumination Equation:
     - C = Cl * max(0, (E * R)^p)
     - E: Eye direction, R: Reflected light direction, p: Specular power.
2. Reflection Vector Calculation
   - Reflection Vector Calculation:
     - R = L - 2 * N(N * L)
     - R: Reflected light direction, L: Light direction, N: Surface normal.
3. Blinn Half-Angle Model
   - Halfway Vector Calculation:
     - H = (L + E) / |L + E|
     - H: Halfway vector, L: Light direction, E: Eye direction.
   - Blinn Half-Angle Equation:
     - C = Cl * (H * N)^p
     - p: Specular power.
4. Shading Equation for Diffuse and Specular Reflections
   - Combined Shading Equation:
     - C = Cr * (Ca + Cl * max(0, N * L)) + Cl * Cp * (H * N)^p
     - Cr: Diffuse color of the surface.
     - Ca: Ambient light color.
     - Cl: Color of the light.
     - Cp: Color of the specular highlight.
     - p: Specular exponent.
5. Applying Shading Models
   - Shading Options:
     - Per-polygon shading (flat shading).
     - Per-vertex shading (Gouraud interpolation).
     - Per-pixel shading (Phong interpolation).
   - Per-Vertex Shading:

        - Surface normal calculated at each vertex.
        - Shading equation applied to each vertex.
        - Color interpolated for each pixel based on vertex colors.
6. Conclusion
    - Explored specular reflection models: Phong and Blinn Half-Angle.
    - Defined reflection vector and halfway vector for specular highlights.
    - Detailed shading equation considering diffuse and specular reflections.
    - Discussed shading options for smooth rendering: per-polygon, per-vertex, and per-pixel.

Gouraud Interpolation, Human Vision, and Color Representation

1. Gouraud Interpolation
    - Calculating Normal Vectors:
        - Normal vectors calculated for each vertex.
        - Normal calculation methods include averaging neighboring polygon normals.
    - Interpolation for Color:
        - Linear interpolation of color along polygon edges.
        - Smoothens edges but may cause blurriness.
2. Phong Interpolation
    - Per-Pixel Shading:
        - Interpolation of surface normals across the polygon for each pixel.
        - Allows precise shading and specular highlights.
    - Advantages Over Gouraud:
        - Superior shading, especially for highlights.
        - Accurately represents sharp highlights within polygons.
3. Human Vision and Color
    - Visible Light Spectrum:
        - Humans perceive light in the ~380nm-700nm wavelength range.
        - Violet (shorter wavelengths) to red (longer wavelengths) and intermediate colors.
        - Ultraviolet waves, x-rays, gamma rays (shorter), infrared, radio waves (longer).
    - Eye Structure:
        - Lens, cornea, and iris focus and regulate light.
        - Retina with rods (brightness) and cones (color).
        - Fovea has high cone density for color sensitivity.
        - Blind spot where optic nerve exits retina.
    - Color Receptors:
        - Three types of cones: short (blue-sensitive), medium (green-sensitive), long (red-sensitive).
        - Overlapping sensitivity ranges for color perception.
4. Color Representation
    - Trichromatic Color Vision:
        - Human vision primarily based on three colors: red, green, blue.
        - Matrix representation for colors using these three fundamental colors.

- Variability in Color Vision:
    - Color vision varies across species (e.g., Mantis Shrimp with extensive color range).
    - Human variability, color blindness, and tetrachromacy.
- CIE Chromaticity Diagram:
    - Represents colors visible to most people.
    - Composite colors, not strictly tied to individual wavelengths.
- Mantis Shrimp:
    - Example of an animal with extensive color vision.

Color Perception, CIE Chromaticity Diagram, Color Models

1. Color Perception and Cones in the Human Eye
    - Blue cone receptors are fewer than green/red, influencing color perception.
    - Overlapping sensitivity ranges of cones lead to color perception continuity.
2. CIE Chromaticity Diagram
    - 3D representation using X, Y, Z axes (X: red-green, Y: blue-yellow, Z: brightness).
    - Curved region represents visible chromaticity values (associated with actual wavelengths).
    - Flat part represents non-spectral colors (mixes without corresponding wavelengths).
    - Complementary colors are opposite on the diagram (e.g., blue-yellow, green-magenta).
3. Display Gamut and Color Representation
    - Gamut: Range of colors a display device can show (usually a triangle within CIE diagram).
    - RGB Color Space: Primary colors (red, green, blue) added to create a wide range of hues.
    - Subtractive Color: Cyan, magenta, yellow; absorb light to create colors.
4. RGB Color Model
    - Colors represented in a Venn diagram with overlapping circles (primary and secondary colors).
    - Mixes of primary colors yield other colors, including white and black.
    - Representation as an RGB color cube from (0,0,0) to (1,1,1) for practical use.
5. Subtractive Color Model (CMYK)
    - Primary colors: Cyan, magenta, yellow, with the addition of black for richer blacks.
    - Mixing these colors yields different colors (e.g., cyan + magenta = blue).
6. Metamers and Color Perception
    - Metamers: Different spectral distributions that appear the same to humans.
    - Spectrally pure colors obtained directly on the spectrum (single-wavelength photons).
    - Metameric colors result from a mix of different wavelengths.
7. Conclusion
    - Explained color perception, cones, and their influence on color understanding.
    - Introduced the CIE chromaticity diagram and its significance.
    - Discussed display gamut, color models (RGB, CMYK), and metameric colors.
    - Previewed exploration of other color spaces for ease of human design.

Color Spaces and Ray Tracing

1. Color Spaces - HSV and HSL
    - HSV (Hue-Value-Saturation):
        - Hue represents the actual color (e.g., red, yellow, green).
        - Value corresponds to brightness (from 0 for black to 1 for full brightness).
        - Saturation determines the vibrancy of the color (0 for white/gray to 1 for fully vibrant color).
        - Visualization as a cone with hexagonal base with each hue at one of the corners, enabling easier color selection.
    - HSL (Hue-Saturation-Lightness):
        - Variant of HSV where white is raised to form a double cone, treating white differently.
        - White is positioned above the other colors, showcasing a double-cone structure.
2. Ray Tracing - Basics and Techniques
    - Ray tracing and rasterization are main 3D graphics rendering techniques.
    - Rasterization:
        - Fast and hardware-accelerated, but requires tricks for realistic appearance.
        - Z-buffering is a common approach.
    - Ray Tracing:
        - Slower and computationally intensive but offers realistic visuals without needing extensive tricks.
        - Used significantly in movies and special effects.
        - Ray tracing involves shooting rays from the camera and determining the objects they intersect with to compute pixel colors.
3. Ray Tracing - Ray Definition and Intersection
    - Ray described parametrically as a vector equation: "r(t) = o + t * d".
        - "o" is the origin, and "d" is the direction of the ray.
    - Object surfaces often defined by implicit equations (e.g., spheres).
        - Intersections computed by solving these equations for the ray.

Ray Tracing and Intersection

1. Ray Tracing and Usage in Animation Studios
    - Ray tracing is popular among animators for realistic visuals.
    - Usage by animation studios:
        - Blue Sky Animation (Ice Age movies) - early adoption.
        - Pixar - prominent use from "Monsters University" (2013).
        - Special effects companies since ~2005 widely adopt ray tracing.
2. Surfaces and Ray-Surface Intersection
    - Surfaces defined using implicit equations to determine ray collisions.
    - Example: Unit sphere defined by x^2 + y^2 + z^2 = 1.
        - Intersection calculation with a ray of direction (dx, dy, dz) and length "t."
    - General sphere with radius "r," centered at (xc, yc, zc), defined as:

        - (x - xc)^2 + (y - yc)^2 + (z - zc)^2 = r^2.
3. Ray-Polygon Intersection
    - Ray intersects any polygon using 2D projection onto the polygon's plane.
    - Steps to calculate intersection:
        Calculate ray's intersection with the infinite 2D plane containing the polygon.
        Project the triangle/polygon and the intersection point onto the 2D plane.
        Employ "point-in-polygon" test to determine if the intersection point is inside the polygon.
4. Intersection of Ray with 3D Plane
    - Plane defined implicitly in 3D: ax + by + cz + d = 0.
        - Calculation of ray-plane intersection.
        - Preventing division by zero in the computation.
5. Point-in-Polygon Test
    - Different methods for determining if a point is inside a polygon:
        - Crossing test.
        - Winding method.
        - Professor Turk's recommended approach: the half-plane test.

Ray Tracing and Intersection

1. Half-Plane Test and Line Equation
    - Half-plane test for point inclusion in a triangle.
    - Line equation: f(x,y) = ax + by + c = 0.
    - Calculation of a, b, and c for a line given two points.
2. Plane Equation and Normal Vector
    - Plane equation: f(x,y,z) = ax + by + cz + d = 0.
    - Calculation of normal vector for a plane using three points.
    - Obtaining coefficients a, b, c, and d for the plane equation.
3. Utilizing Half-Plane Test
    - Applying the half-plane test to determine point inclusion in a 2D triangle.
    - Use of line equations for the half-plane test.
    - Example code for implementing the half-plane test.
4. Coordinate Mapping and Eye Rays
    - Mapping between 2D screen coordinates and 3D view plane coordinates.
    - Calculating eye rays for perspective projection in a 3D scene.
    - Field of view calculation based on camera specifications.
5. Reflections and Recursion
    - Reflecting rays for mirror-like surfaces.
    - Calculating color using reflection coefficients and recursive ray tracing.
    - Determining recursion termination based on depth or contribution threshold.
6. Ray Tracing vs. Real-World Light Paths
    - Contrast between ray tracing and real-world light behavior.
    - Discussion on light ray tracing direction.
7. Transparent Surfaces and Refraction

- Understanding light behavior in transparent materials.
- Refraction and index of refraction (IOR) for bending light.
- Common index of refraction values for various materials.

Transparent Surfaces, Shadows, and Effects

1. Transparent Surfaces and Refraction
    - Light transmission and bending in transparent materials.
    - Snell's Law for calculating the bend angle based on IOR.
    - Total internal reflection and its applications.
2. Color Calculation with Transparency
    - Updating color equation for eye ray considering transparency.
    - Including reflection and transmission components in color calculation.
3. Shadows - Hard and Soft Shadows
    - Explanation of hard and soft shadows in ray tracing.
    - Shadow rays and their importance in determining light visibility.
    - Calculation of light contribution considering shadows.
4. Types of Shadows
    - Hard shadows with clear edges.
    - Soft shadows with gradual intensity variations.
    - Calculation of light contribution in shadows using the visibility function.
5. Distribution Ray Tracing for Shadows
    - Exploring distribution ray tracing for soft shadows.
    - Sampling multiple shadow rays for area lights.
    - Averaging shadow visibilities to simulate penumbra.
6. Efficiency and Realism in Shadows
    - Comparison of rendering speed between hard and soft shadows.
    - The realism and visual appeal of soft shadows.
7. Distribution Ray Tracing for Reflections
    - Utilizing distribution ray tracing for glossy reflections.
    - Averaging multiple reflection rays to simulate reflection scattering.
8. Distribution Ray Tracing for Motion Blur
    - Exploring motion blur through time-based ray distribution.
    - Averaging colors across multiple time instances for motion blur effect.

Raytracing Optimization Techniques

1. Introduction to Raytracing Optimization
    - Importance of optimizing raytracing for complex scenes.
    - Goal: Enhance ray-object intersection efficiency.
    - Overview of approaches to improve rendering speed.
2. Bounding Volumes for Speeding Up Raytracing
    - Utilizing bounding boxes to accelerate intersection checks.
    - Advantages of bounding volumes: faster computation and filtering.

- Generalization to other bounding volumes (e.g., spheres, ellipsoids).

3. Bounding Hierarchies for Complex Objects
   - Constructing bounding hierarchies for grouping complex objects.
   - Designing parent-child relationships within the hierarchy.
   - Speeding up ray-object intersection through bounding hierarchy traversal.
4. Strategies for Constructing Bounding Hierarchies
   - Bottom-up vs. top-down approaches for constructing bounding hierarchies.
   - Bottom-up: Pairwise grouping of objects and formation of bounding boxes.
   - Top-down: Recursive partitioning of space into smaller bounding volumes.
5. Evaluation of Bounding Hierarchy Methods
   - Analysis of pros and cons for bottom-up and top-down approaches.
   - Considerations for practical application and efficiency.
   - Balance between accuracy and computational cost.
6. Additional Methods for Raytracing Optimization
   - Overview of other techniques, such as GRIDS and K-D trees.
   - GRIDS method: Spatial partitioning of the scene into a 3D grid.
   - K-D trees: Spatial subdivision for improved intersection queries.
7. Advancements in Raytracing Speed
   - Recognizing the increased efficiency of raytracing over time.
   - Dominance of raytracing in the special effects industry.
   - Prospects for further optimization and performance improvements.

Rasterization Techniques for Shadows and Textures

1. Introduction and Transition from Raytracing to Rasterization
   - Shift from raytracing to rasterization for rendering techniques.
   - Focus on achieving effects like shadows and textures in rasterization.
   - Overview of two primary raster-based shadow creation methods: shadow volumes and shadow mapping.
2. Handling Shadows in Rasterization
   - Understanding shadows in terms of visibility and light-source interactions.
   - Differentiating between attached and cast shadows based on object properties.
   - Introduction to Z-Buffer for handling visibility and shadow calculations.
3. Shadow Mapping Algorithm - Two-Pass Z-Buffer Method
   - Detailed steps of the shadow mapping technique.
   - Rendering the scene from the light source's perspective and then from the camera's perspective.
   - Mapping hidden pixels in light space to shadows in the final render using the Z-buffer.
4. Addressing Efficiency and Performance in Shadow Generation
   - Recognizing the computational cost of shadow creation for each light source.
   - Balancing between rendering multiple light sources and rendering speed.
5. Texture Mapping in Rasterization
   - Mapping textures onto 3D objects using texture coordinates.
   - Interpolating texture coordinates across polygons during rasterization.

- Texture lookup and color assignment based on texture color for each pixel.

6. Texture Mapping Process - Sub-Stages
    - Exploring the first sub-stage of texture mapping: interpolation of texture coordinates.
    - Handling texture coordinates (S, T) and understanding their role in texture mapping.
    - Discussing challenges related to perspective projection in texture coordinate interpolation.