Review: N2D Transformation

- The N2D transformation is used to change the coordinate system from view coordinates to device coordinates.
- This transformation is used to project vertices from the view coordinate system to the device coordinate system.
- The equation for the N2D transformation is:

\[
p = \begin{bmatrix}
    x'
    
y'
    
z'
    
    1
\end{bmatrix} = \begin{bmatrix}
    x
    
y
    z
    
    1
\end{bmatrix}
\begin{bmatrix}
    -d
    
    0
    
    0
    
    1
\end{bmatrix}
\begin{bmatrix}
    1
    
    0
    
    0
    
    0
\end{bmatrix}
\begin{bmatrix}
    0
    
    d
    
    0
    
    1
\end{bmatrix}
\begin{bmatrix}
    0
    
    0
    
    0
    
    d
\end{bmatrix}
\begin{bmatrix}
    0
    
    0
    
    0
    
    1
\end{bmatrix}
\begin{bmatrix}
    0
    
    0
    
    0
    
    1
\end{bmatrix}
\begin{bmatrix}
    0
    
    0
    
    0
    
    1
\end{bmatrix}
\begin{bmatrix}
    0
    
    0
    
    0
    
    1
\end{bmatrix}
\begin{bmatrix}
    1
    
    0
    
    0
    
    0
\end{bmatrix}
\begin{bmatrix}
    1
    
    0
    
    0
    
    0
\end{bmatrix}
\begin{bmatrix}
    0
    
    0
    
    0
    
    1
\end{bmatrix}
\begin{bmatrix}
    0
    
    0
    
    0
    
    1
\end{bmatrix}
\begin{bmatrix}
    0
    
    0
    
    0
    
    1
\end{bmatrix}
\begin{bmatrix}
    0
    
    0
    
    0
    
    1
\end{bmatrix}
\end{bmatrix}
\begin{bmatrix}
    x
    
y
    z
    
    1
\end{bmatrix}

Display z range is 0 to 1.

glDepthRange(n,f) can constrain further, but depth = 1 is both max and default.

Review: Projective Rendering Pipeline

- The projective rendering pipeline is a series of steps that transform vertices from the object world coordinate system to the device coordinate system.
- The pipeline consists of the following steps:
  1. Modeling transformation
  2. Viewing transformation
  3. Perspective transformation
  4. Projecting transformation
  5. Clipping transformation

- The pipeline is read from top to bottom, where the object is transformed and then projected to the device coordinate system.

Review: WebGL Example

- The WebGL example is a demonstration of how to use WebGL to render a 3D scene.
- The example code is provided in the document.
- The code includes the necessary steps to create a 3D scene and render it using WebGL.

Review: Coord Sys: Frame vs Point

- The document provides an example of how to use WebGL to render a 3D scene.
- The example code is provided in the document.
- The code includes the necessary steps to create a 3D scene and render it using WebGL.

OPENGL RENDERING PIPELINE

- The OpenGL rendering pipeline is a series of steps that transform vertices from the object world coordinate system to the device coordinate system.
- The pipeline consists of the following steps:
  1. Scene
  2. Camera Coords
  3. Device Coords
  4. Vertex Shader
  5. Fragment Shader
  6. Pre-Service Operations
  7. Image

VERTEX SHADER

- The vertex shader is a program that processes vertices in the scene.
- The vertex shader is used to transform vertices from the object world coordinate system to the device coordinate system.
- The vertex shader includes the following steps:
  1. Uniform variables
  2. Varying variables
  3. Fragment shader
  4. Screen color
  5. Frame buffer
Review: Light Sources
• directional/parallel lights
  • point at infinity: \((x,y,2,0)^T\)
• point lights
  • finite position: \((x,y,2,1)^T\)
• spotlights
  • position, direction, angle
• ambient lights

Review: Light Source Placement
• geometry: positions and directions
  • standard: world coordinate system
  • effect: lights fixed wrt world geometry
  • alternative: camera coordinate system
  • effect: lights attached to camera (car headlights)

Review: Light Source Placement
• directional/parallel lights
• point lights
• spotlights
• ambient lights

Review: Shading Models
• flat shading
  • for each polygon
  • compute Phong lighting just once
• Gouraud shading
  • compute Phong lighting at the vertices
  • for each pixel in polygon, interpolate colors
• Phong shading
  • for each pixel in polygon
  • interpolate normal
  • compute Phong lighting

Review: Non-Photorealistic Shading
• cool-to-warm shading: \(k_r = \frac{1 + (n \cdot l)}{2} - k_r \cdot (1 - k_r \cdot l)\)
• draw silhouettes: if \((n \cdot l) \leq 0\), edge-eye vector
• draw creases: if \((n \cdot l) < \text{threshold}\)

Review: Texture Coordinates
• texture image: 2D array of color values (texels)
• assigning texture coordinates \((u,v)\) at vertex with object coordinates \((x,y,z,w)\)
  • sometimes called \(((u,v)\) instead of \((x,y)\)
  • use interpolated \((u,v)\) for texel lookup at each pixel
  • use value to modify a polygon color or other property
  • specified by programmer or artist

Review: Texture Coordinates
• texture image: 2D array of color values (texels)
• assigning texture coordinates \((u,v)\) at vertex with object coordinates \((x,y,z,1)\)
  • use \((u,v)\) for texel lookup at each pixel
  • use value to modify a polygon color or other property
  • specified by programmer or artist

Review: Tiled Texture Map

Review: Fractional Texture Coordinates

Ray Tracing
Review: Recursive Ray Tracing
- ray tracing can handle
  - reflection (chrome/mirror)
  - refraction (glass)
  - shadow
- one primary ray per pixel
- spawn secondary rays
  - reflection, refraction
  - if another object is hit, recurse to find its color
- shadow
- cast ray from intersection point to
  - surface
  - check if intersects another object
- termination criteria
  - no intersection (ray exits scene)
  - max bounces (recursion depth)
  - attenuated below threshold

Review: Reflection and Refraction
- reflection: mirror effects
  - perfect specular reflection
- refraction: at boundary
  - Snell’s Law
  - light ray bends based on refractive indices $c_1$, $c_2$
  - $c_1 \sin \theta_1 = c_2 \sin \theta_2$

Review: Ray Tracing
- issues:
  - generation of rays
  - intersection of rays with geometric primitives
  - geometric transformations
  - lighting and shading
- efficient data structures so we don’t have to test intersection with every object

Review: Ray-Sphere Intersections, Lighting
- Intersections: solving a set of equations
  - Using implicit formulas for primitives
- Direct illumination: gradient of implicit surface

Example: Ray-Sphere intersection
$$r_1(x_1, y_1, z_1) = \sqrt{x_1^2 + y_1^2 + z_1^2}$$
$$r_2(x_2, y_2, z_2) = \sqrt{x_2^2 + y_2^2 + z_2^2}$$

Example: Sphere normals
$$(x, y, z) \cdot \nabla r_1 = 0$$

Review: Procedural/Collision
- Procedural/Collision
  - textures, geometry
  - procedural approach
    - compute something on the fly
      - load from disk
      - often less memory cost
      - visual richness
      - adaptable precision
- noise, fractals, particle systems

Review: Fractal Terrain
- 1D: midpoint displacement
  - divide in half, randomly displace
  - scale variance by half
- 2D: diamond-square
  - generate new value at midpoint
  - average corner values + random displacement
  - scale variance by half each time

Review: Particle Systems
- changeable/flow stuff
  - fire, steam, smoke, water, grass, hair, dust, waterfalls, fireworks, explosions, flocks
- life cycle
  - generation, dynamics, death
- rendering tricks
  - avoid hidden surface computations

Review: Collision Detection
- boundary check
  - perimeter of world vs. viewpoint or objects
  - 20/30 absolute coordinates for bounds
  - simple point in space for viewpoint/objects
- set of fixed barriers
  - walls in maze game
  - 20/30 absolute coordinate system
- set of moveable objects
  - one object against set of items
  - missile vs. several tanks
  - multiple objects against each other
  - punching game: arms and legs of players
  - room of bouncing balls

Review: Collision Proxy Tradeoffs
- collision proxy (bounding volume) is piece of geometry used to represent complex object for purposes of finding collision
- proxies exploit facts about human perception
  - we are bad at determining collision correctness
  - especially many things happening quickly
- increasing complexity & tightness of fit decreasing cost of (overlap tests + proxy update)

Review: Z-Buffer Algorithm
- augment color framebuffer with Z-buffer or
  - depth buffer which stores Z value at each pixel
  - at frame beginning, initialize all pixel depths to $\infty$
  - when rasterizing, interpolate depth (Z) across polygon
  - check Z-buffer before storing pixel color in
    - framebuffer and storing depth in Z-buffer
  - don’t write pixel if its Z value is more distant than the Z value already stored there

Review: Depth Test Precision
- reminder: perspective transformation maps eye-space (VCS) z to NDC z
- $E = \frac{x + A}{z}$
- $F = \frac{y + B}{z}$
- $C = \frac{z + D}{w}$
- $D = z + w$
- $z_{eye} = \left( C + D \right) \frac{1}{w}$
- thus: depth buffer essentially stores $1/z$ (for VCS z)
  - high precision for near, low precision for distant

Review: Hidden Surfaces / Picking / Blending
- Hidden Surfaces / Picking / Blending
- uniform grids
- BSP trees
- kd-trees
- octrees
- OBB Trees

Backstory: 2D Parametric Lines
- $p(t) = p_0 + t(p_1 - p_0)$
- $p_0$ at start point
  - $p_1$ go towards $p_0$
  - according to parameter $t$
  - $p(0) = p_0, p(1) = p_1$

Review: Language-Based Generation
- L-Systems
  - F: forward, R: right, L: left
  - Koch snowflake:
    - $F = \text{FLRFRLF}$
    - Mariano’s Bush:
      - $F=F(-F+F+F-F)+[+F-F-F]$ (+ angle 16)

Backstory: 2D Parametric Lines
- $p(t) = p_0 + t(p_1 - p_0)$
- $p_0$ at start point
  - $p_1$ go towards $p_0$
  - according to parameter $t$
  - $p(0) = p_0, p(1) = p_1$

Review: Depth Test Precision
- reminder: perspective transformation maps eye-space (VCS) z to NDC z
- $E = \frac{x + A}{z}$
- $F = \frac{y + B}{z}$
- $C = \frac{z + D}{w}$
- $D = z + w$
- $z_{eye} = \left( C + D \right) \frac{1}{w}$
- thus: depth buffer essentially stores $1/z$ (for VCS z)
  - high precision for near, low precision for distant
Review: Integer Depth Buffer
- reminder from viewing discussion: depth ranges
  - VCS range [0.5, 1], DCS range [0.1, 0.5]
  - convert fractional real number to integer format
    - multiply by 2^n then round to nearest int
      - where n = number of bits in depth buffer
  - 24 bit depth buffer = 2^24 = 16,777,216 possible values
    - small numbers near, large numbers far
  - consider VCS depth: \( z_{\text{VCS}} = [3<<N] \times (a + b / z_{\text{VCS}}) \)
    - \( N = \) number of bits of Z precision, \( 1<<N \) bitshift = 2^n
  - \( a = z_{\text{Far}} / (z_{\text{Near}} - z_{\text{Near}}) \)
  - \( b = z_{\text{Far}} / (z_{\text{Near}} - z_{\text{Far}}) \)
  - \( z_{\text{VCS}} \) = distance from the eye to the object
Full derivation at https://www.opengl.org/archives/resources/faq/technical/depthbuffer.htm

Review: Picking Methods
- raycaster intersection support
- offscreen buffer color coding

Review: Object Space Algorithms
- determine visibility on object or polygon level
  - using camera coordinates
  - resolution independent
  - explicitly compute visible portions of polygons
  - early in pipeline
    - after clipping
    - requires depth-sorting
      - painter’s algorithm
      - BSP trees

Review: Image Space Algorithms
- perform visibility test for in screen coordinates
  - limited to resolution of display
  - Z-buffer: check every pixel independently
  - performed late in rendering pipeline

Review: Back-face Culling
- \( z_{\text{Far}} = \) distance from the eye to the object
- \( z_{\text{Near}} = \) distance from the eye to the object

Review: Invisible Primitives
- why might a polygon be invisible?
  - polygon outside the field of view / frustum
    - solved by clipping
  - polygon is back-facing
    - solved by backface culling
  - polygon is occluded by object(s) nearer the viewpoint
    - solved by hidden surface removal

Review: Painter’s Algorithm
- draw objects from back to front
- problems: no valid visibility order for
  - intersecting polygons
  - cycles of non-intersecting polygons possible
  - solved by hidden surface removal
  - BSP trees

Review: Blending with Premultiplied Alpha
- specify opacity with alpha channel:
  - 0 = transparent, 255 = opaque
- how to express a pixel that is covered by a red object?
  - obvious way: store color independent from transparency (r,g,b,a)
    - intuation = alpha as transparent colored glass
    - 0% transparency can be represented with many different RGB colors
    - pixel value is \((1,1,1,1)\)
  - upside: easy way to change opacity of image, very intuitive
  - downside: computing calculations are more difficult - not associative
    - elegant way: premultiply by a color matrix (e.g., \( \alpha \tau \))
    - intuation = alpha as screen/mesh
      - RGB specifies how much color each pixel contributes to curve
    - alpha specifies how much object obscures whatever is behind it (coverage)
    - upside: \( \alpha \times \tau \) maps half the pixel is covered by the object, half is fully transparent
      - only one channel represents 100% transparency (0,0,0,0)
    - downside: less intuitive

Review: Metamers
- brain sees only cone response
  - different spectra appear the same: metamers

Review: Measured vs. CIE XYZ Color Spaces
- measured basis:
  - monochromatic lights
  - physical observations
  - negative sides
- CIE standard, all positive
- measured basis:
  - “natural” lights
  - all positive, unit area
  - \( Y \) is luminance

Review: Spectral Sensitivity Curve
- plane of equal brightness showing chromaticity
- gamut is polygon, device primaries at corners
  - defines reproducible color range

Backstory: Spectral Sensitivity Curve

Backstory & Review: Trichromacy
- trichromacy
  - three types of cones: S, M, L
  - color is combination of cone stimuli
    - different cone responses: area function of wavelength
  - for a given spectrum
    - multiply by responses curve
    - integrate to get response

Backstory: Spectral Sensitivity Curve

Review: Color Constancy
- automatic "white balance" from change in illumination
- vast amount of processing behind the scenes!
- colorimetry vs. perception

Review: HSI/HSV and RGB
- HSI/HSV conversion from RGB
  - hue same in both
  - value is max, intensity is average
  \[ H = \cos^{-1}\left(\frac{1}{2}(R - G) + (R - B)\right) \]
  \[ I = \frac{H + G - B}{3} \]
- HSV:
  \[ S = 1 - \min(R,G,B) \]
  \[ V = \max(R,G,B) \]

Review: YIQ Color Space
- color model used for color TV
  - Y is luminance (same as CIE)
  - I & Q are color (not same as HSI)
  - using Y backwards compatible for B/W TVs
  - conversion from RGB is linear
  \[
  \begin{align*}
  Y &= 0.30 \cdot R + 0.59 \cdot G + 0.11 \cdot B \\
  I &= 0.21 \cdot R - 0.72 \cdot G - 0.11 \cdot B \\
  Q &= 0.29 \cdot R + 0.58 \cdot G + 0.11 \cdot B
  \end{align*}
  \]
- green is much lighter than red, and red lighter than blue

Review: Channel Rankings
- expressiveness principle
  - match channel and data characteristics
- effectiveness principle
  - encode most important attributes with highest ranked channels

Review: Marks and Channels
- marks
  - geometric primitives
  - position
  - color
- channels
  - control appearance of marks

Review: Luminance vs. Intensity
- luminance
  - Y of YIQ
  - 0.299R + 0.587G + 0.114B
- captures important factor
- intensity/value/brightness
  - I/V/B of HSI/HSV/HSB
  - 0.333R + 0.333G + 0.333B
- not perceptually based

Review: RGB Color Space (Color Cube)
- define colors with (r, g, b) amounts of red, green, and blue
  - used by OpenGL
  - hardware-centric
- RGB color cube sits within CIE color space
  - subset of perceivable colors
  - scale, rotate, shear cube

Review: Blackbody Curve
- blackbody curve
  - combustion
  - candle 2000K
- daylight 6500K
- overcast day 7000K
- lightning >20,000K

Review: YIQ Color Space
- Y is luminance (same as CIE)
- I & Q are color (not same as HSI)
- using Y backwards compatible for B/W TVs
- conversion from RGB is linear

Review: HSV Color Space
- hue: dominant wavelength, "color"
- saturation: how far from grey
- value: how far from black/white
  - aka brightness, intensity: HSB/HSV/HIS similar
- cannot convert to RGB with matrix alone

Review: Visualization
- compliment of marks
- control appearance of marks
- position
- color
- identity
- channel
- attribute
- magnitude
- attributes
- length
- area
- volume
- shape
- tilt
- color
- spatial
- hue
- luminance
- saturation
- intensity
- brightness
- expression
- effectiveness
- data
- characteristic
- bin
- bar
- bar chart
- histogram