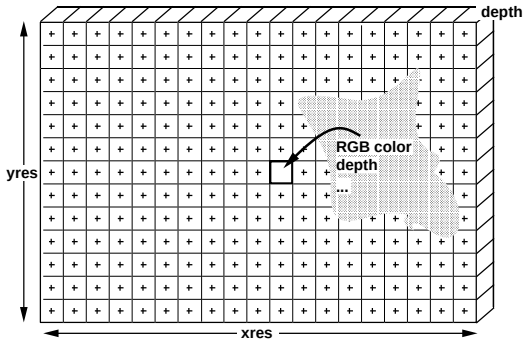
Lecture 7: 30 September 1999

Administrative:
- Asst 3 (object modeling) due tomorrow 5pm
  - `imagery4` flakiness – under investigation
  - In meantime, verify saves, or use other space
  - Make sure to follow `turnin` instructions !
- Asst 4 handout, end of class (plz. remind me)

Today:
- Perspective: qualitative and quantitative (L6)
- Transforming from NDC to Viewport coords
- Polygon Scan Conversion: Preliminaries

Framebuffer and Discrete Coordinates

Usually "RGB" or "24-bit" or "full color" framebuffer
2D array of *xres* × *yres* **pixels** (picture elements)

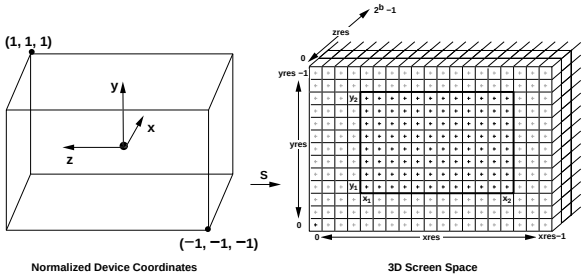


At each pixel can be stored:
- *Color* (3×8 bit RGB = 24 bits, sometimes 36)
- *Depth* ($b = 24$ or $b = 32$ bits: values $0..2^b - 1$)
- Other attributes: α, stencil, window ID, etc.
  - High-end machine may store ~200 bits per pixel!

Today: coordinatizing, filling the framebuffer
- Viewport Transformation
- 3D Screen Space

Discretizing the Normalized View Volume

NDC parallelepiped is $x = \pm 1$, $y = \pm 1$, $z = \pm 1$
Map this parallelepiped into 3D "device coordinates"

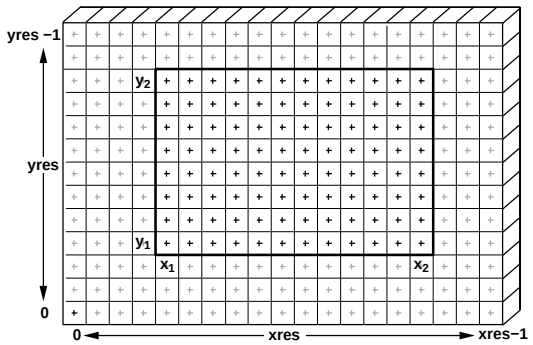


x, y interval:
- resolution depends on framebuffer resolution
- typically specified as a rectangular "viewport":
  - (x_1, y_1) to (x_2, y_2)

z interval:
- resolution depends on h/w precision (*depth buffer*)
- b bits: unsigned integer in range $[0..2^b - 1]$

Viewport Transformation (first attempt)

Integer coordinates at pixel centers, origin at lower left
- Easy: map $x \in [-1.. + 1]$ to $x_S \in [x_1..x_2]$,
- and map $y \in [-1.. + 1]$ to $y_S \in [y_1..y_2]$



$x_S =$ ____

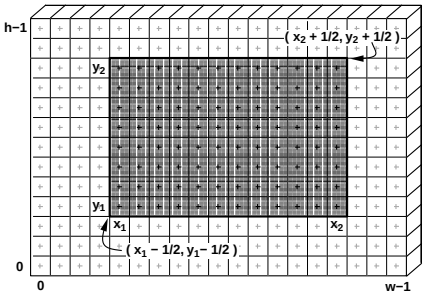
$y_S =$ ____

Viewport Transformation (corrected)

Accounting for pixel extents means mapping:

$x \in [-1.. + 1]$ to $x_S \in [x_1 - \frac{1}{2}..x_2 + \frac{1}{2}]$

$y \in [-1.. + 1]$ to $y_S \in [y_1 - \frac{1}{2}..y_2 + \frac{1}{2}]$



$$x_S = x_1 - 0.5 + \frac{x+1}{2}(x_2 - x_1 + 1)$$

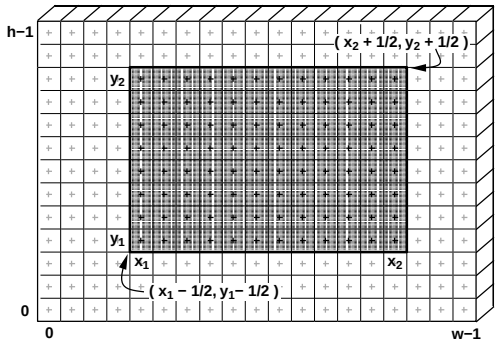
$$y_S = y_1 - 0.5 + \frac{y+1}{2}(y_2 - y_1 + 1)$$

Depth (z) is mapped linearly:

$$z_S = \frac{z+1}{2}(2^b - 1)$$

Viewport Transformation

Matrix formulation



$$\mathbf{S} = \begin{pmatrix} \frac{w}{2} & 0 & 0 & (x_1 + \frac{w-1}{2}) \\ 0 & \frac{h}{2} & 0 & (y_1 + \frac{h-1}{2}) \\ 0 & 0 & \frac{r}{2} & \frac{r}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

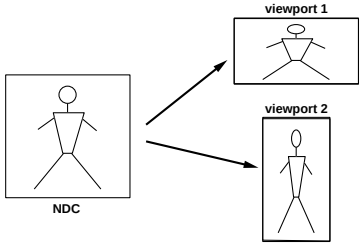
where $w = x_2 - x_1 + 1$, $h = y_2 - y_1 + 1$, $r = 2^b - 1$

$\mathbf{S}$ takes NDC coordinates to 3D Screen Space

Aspect Ratio

Ratio of largest to smallest dimension

Changes iff $(1,1)$, $(2,2)$ entries of $\mathbf{S}$ are unequal

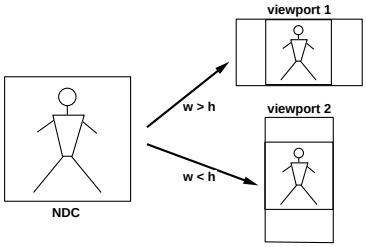


Preserving Aspect Ratio

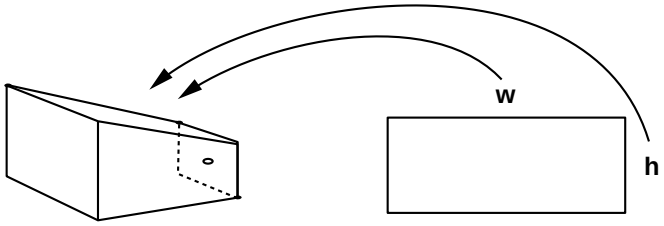
Strategies for preserving aspect ratio:

Assume field of view equal in x, y

Scale NDC into smaller of viewport dimensions



Derive x, y fields of view from viewport width, height



How ?

Pipeline Transformations: Summary

Prepend **S** to existing matrix string:

$$\begin{pmatrix} x_S \\ y_S \\ z_S \end{pmatrix} = \mathbf{S}\ \mathbf{P}\ \mathbf{E}\ \mathbf{M}_O \begin{pmatrix} x_O \\ y_O \\ z_O \\ 1 \end{pmatrix}$$

$\mathbf{M}_O$ Modeling transformations (Object to World)
(One matrix per object)

E Re-express World coordinates in Eye coord. sys.
Translate, then Rotate ($\mathbf{E} = \mathbf{RT}$)

P Perspective transformation into NDC
Device-independent (produces canonical coords.)
Projection from 4- to 3-vector happens after **P**

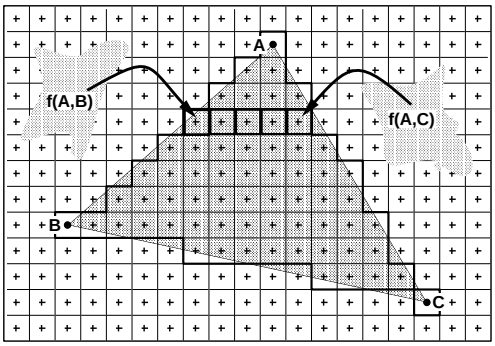
S Transformation into screen coordinates
Device-dependent

Scan-Conversion Preliminaries

3D Polygon Scan Conversion (H&B S3.11, 13.5)
Interpolation of Color, Depth

Conversion of *continuous* primitive
to its discrete *pixel coverage*

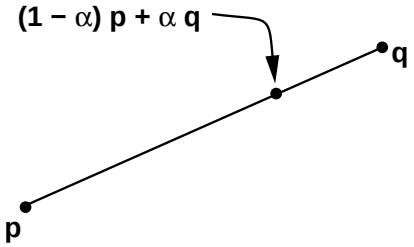
Color (depth, etc.) associated with vertices
interpolate across polygon interior



Step along left, right triangle edges
interpolate color (depth, etc.) at pixels

Linear Interpolation

Consider interpolation along line segment



Coefficients non-negative, sum to 1 ⇒ interpolation
yields a *convex combination* of input samples

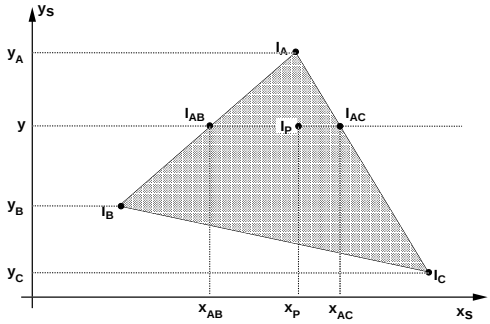
Can interpolate geometry this way:
Points
Vectors (e.g., Normals)

But also:
Colors
Transparency
Texture coordinates
...

Gouraud Interpolation

Depth, color associated with polygon vertices
Depth: z coordinate, from perspective transform
Color: radiometry (shading), from lighting model

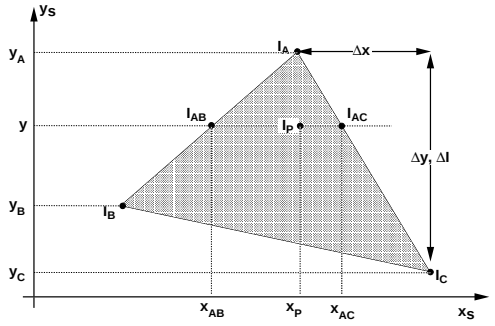
Interpolate polygon edges to find *span* endpoints
interpolate span interiors to find pixel values



Terminology: Shading, Interpolation

Scan Converting One Triangle

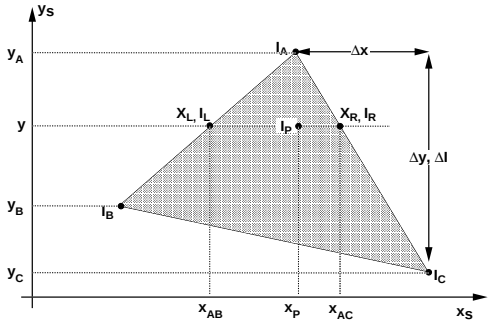
Scanline loop (x, y, intensity I per vertex):
FillPoly ($x_A, y_A, I_A, x_B, y_B, I_B, x_C, y_C, I_C$)



$\Delta X_{AB} = \frac{x_B - x_A}{y_B - y_A}$; $\Delta X_{AC} = \frac{x_C - x_A}{y_C - y_A}$
$\Delta I_{AB} = \frac{I_B - I_A}{y_B - y_A}$; $\Delta I_{AC} = \frac{I_C - I_A}{y_C - y_A}$
$x_{AB} = x_{AC} = x_A$; $I_{AB} = I_{AC} = I_A$
For $y \in y_A..y_B$
  FillSpan$(y, x_{AB}, I_{AB}, x_{AC}, I_{AC})$
  x_{AB} += ΔX_{AB}
  x_{AC} += ΔX_{AC}
  I_{AB} += ΔI_{AB}
  I_{AC} += ΔI_{AC}
For $y \in y_B..y_C$
  ...

Pseudo code

Span filling:



FillSpan(y, x_L, I_L, x_R, I_R)
$\Delta I = \frac{I_R - I_L}{x_R - x_L}$
$I = I_L$
For $x \in x_L..x_R$
  setPixel(x, y, I)
  I += ΔI
Analogous interpolation of depth (not yet needed)

Recap: High-Level Strategy

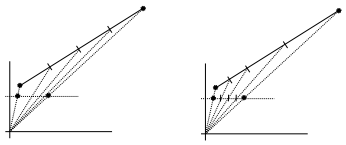
1) Shade vertices
2) Transform, project vertices into screen space
3) Interpolate screen space depth, color

Gouraud Flaw – Convex Interpolation

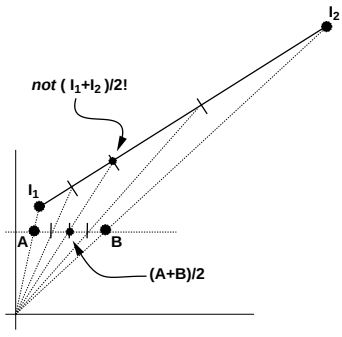
Demo 1:
`SceneViewer gouraudhighlight1.iv -e gouraudhighlight.env`
What's happening?
What if polygon interior has sharp highlight?
Problem? How to fix it.
What if polygon vertex has sharp highlight?
Problem? How to fix it.
Demo 2:
`SceneViewer gouraudhighlight2.iv -e gouraudhighlight.env`

Screen Space Color Interpolation

Perspective transformation does *not* preserve distances!



Interpolation before, after perspective yields different results!
Shading parameters should *not* be interpolated in screen space



Demo:
`SceneViewer gouraudSSinterpolation.iv`