Lecture 9: 3 October 2002

Office hours today:

Prof. T.: after class in 4-035

Addy: 4-6pm in **W20 Linux cluster** (note change)

Today Part I:

Finish polygon scan conversion & filling

Today Part II:

Assignment 2 (`*_scene` exhibition)

Mysteries of homogeneous coordinates explained

Assignment 4 (Polygon fill with `ivscan`) posted

Friday:

Assignment 3 (Polygon wireframe rendering) due 5pm

Assignment 2 Comments

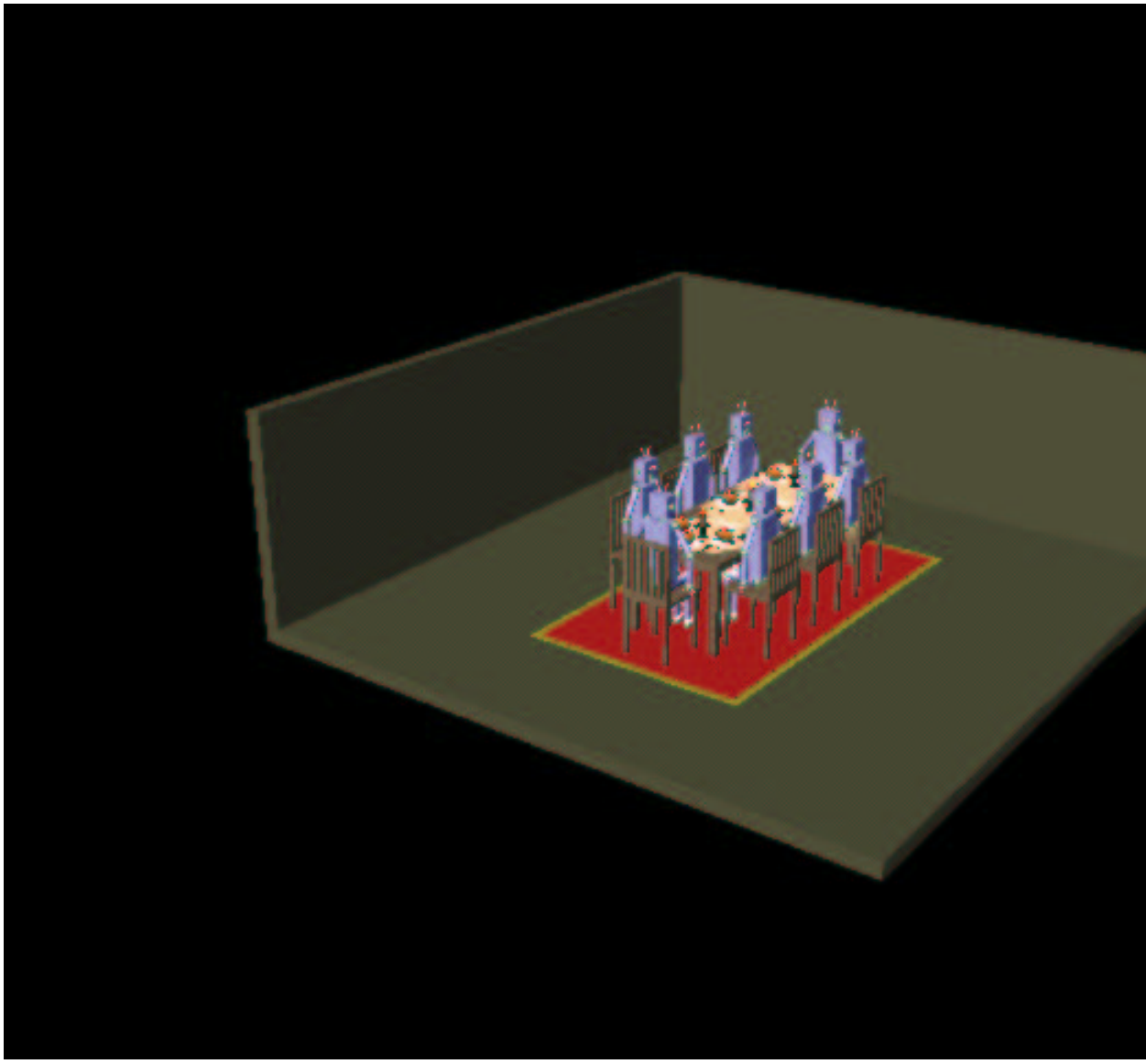
Great work!

- Lots of thought evident in most scene compositions
- Many interesting scenes: fun, serious, political, etc.
- Lots of effort selecting, situating, and lighting objects
- Group spirit of using & acknowledging eachother's work

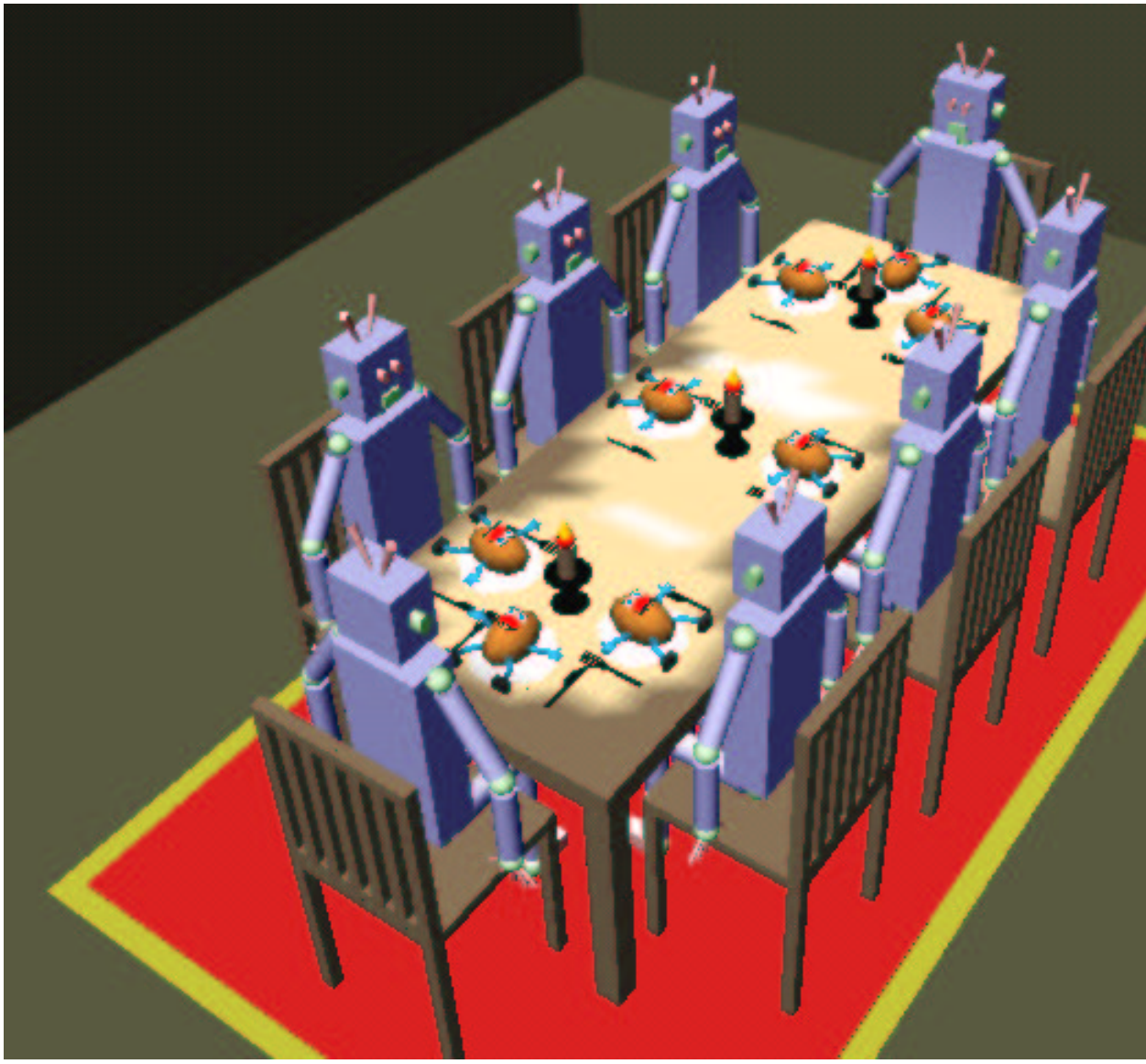
Common technical pitfalls

- Some objects did not respect Ass't 1 modeling conventio
- Slow render times for some scenes:
- Excessive object/polygon budget
- Large numbers of light sources
- Use of software clipping planes
- Not exploiting Separators to "bind" lights only to certai
- Erroneously expecting cast shadows (Inventor/OpenGL

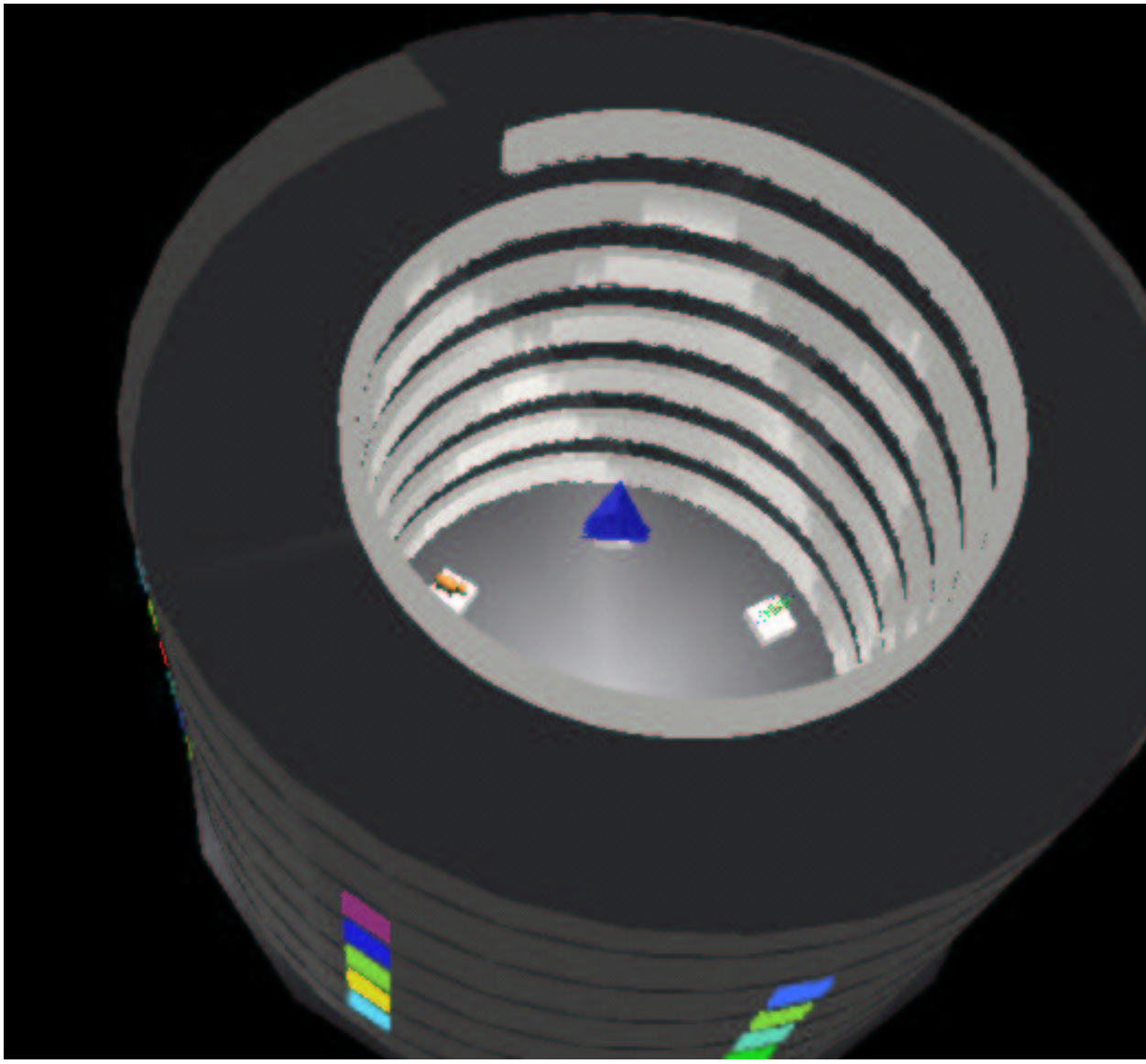
Daniel Chak (chak)



Daniel Chak (chak)

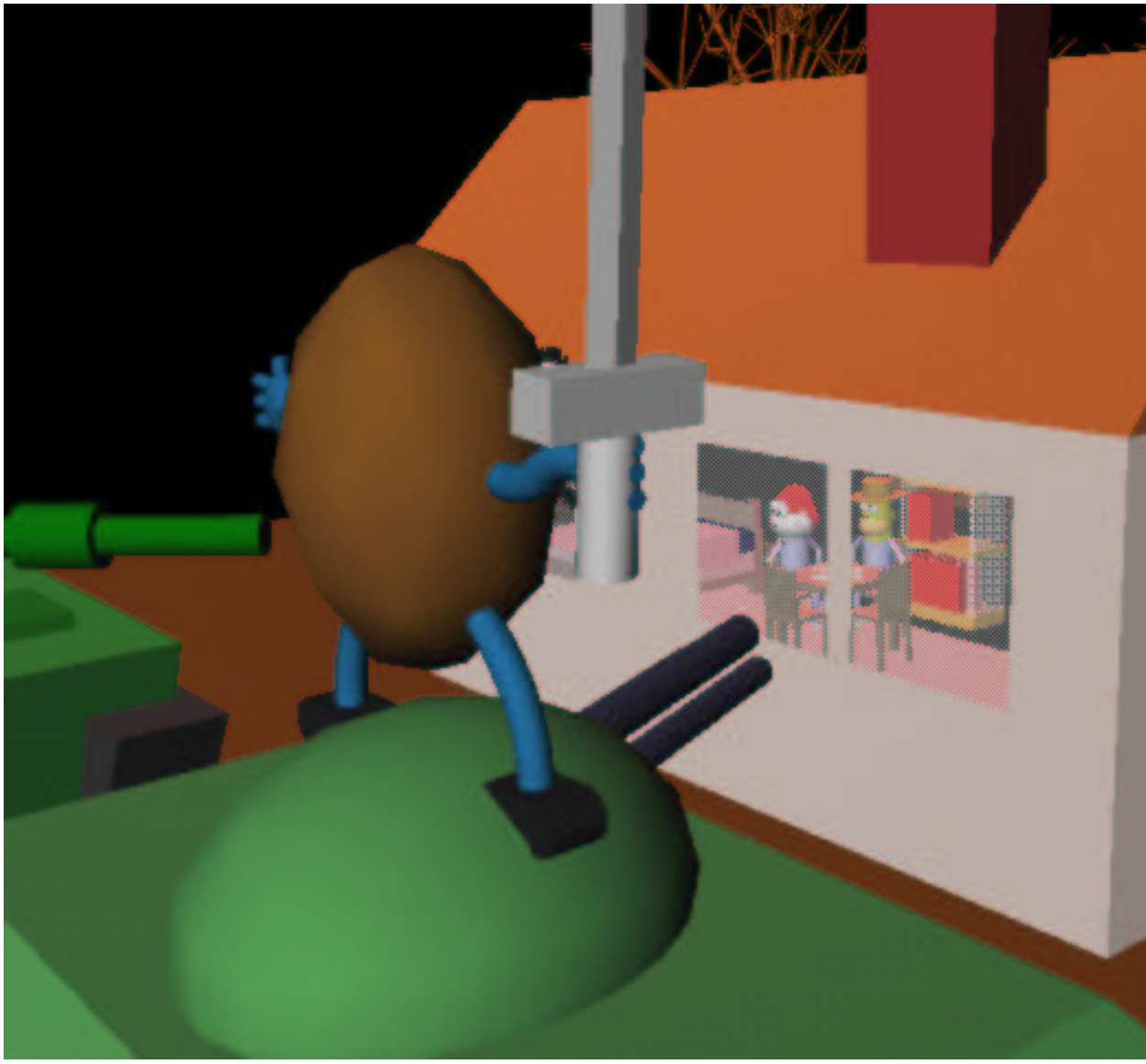


Ying Li (cyli)

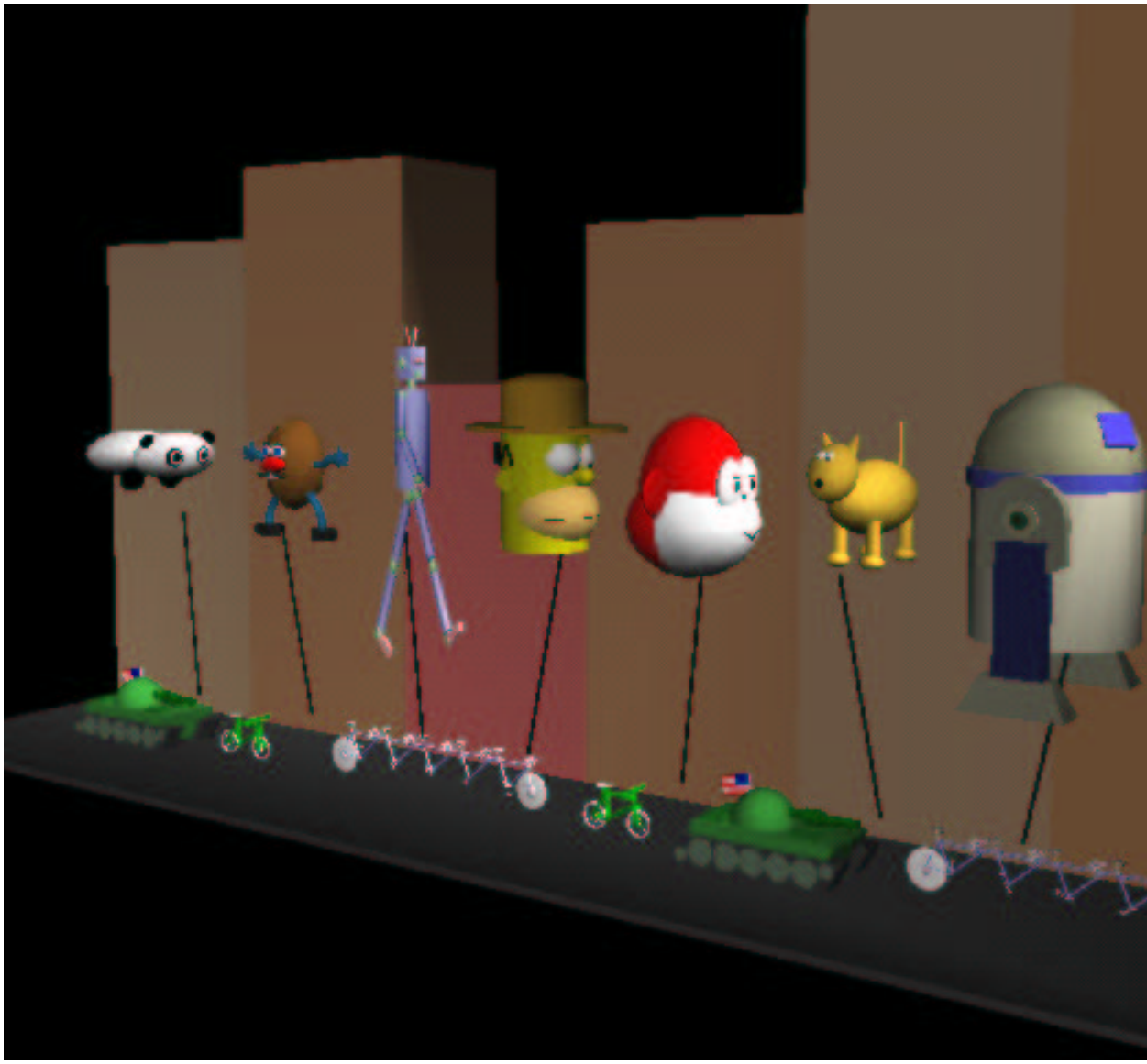


Patrick Menard (dranem05)

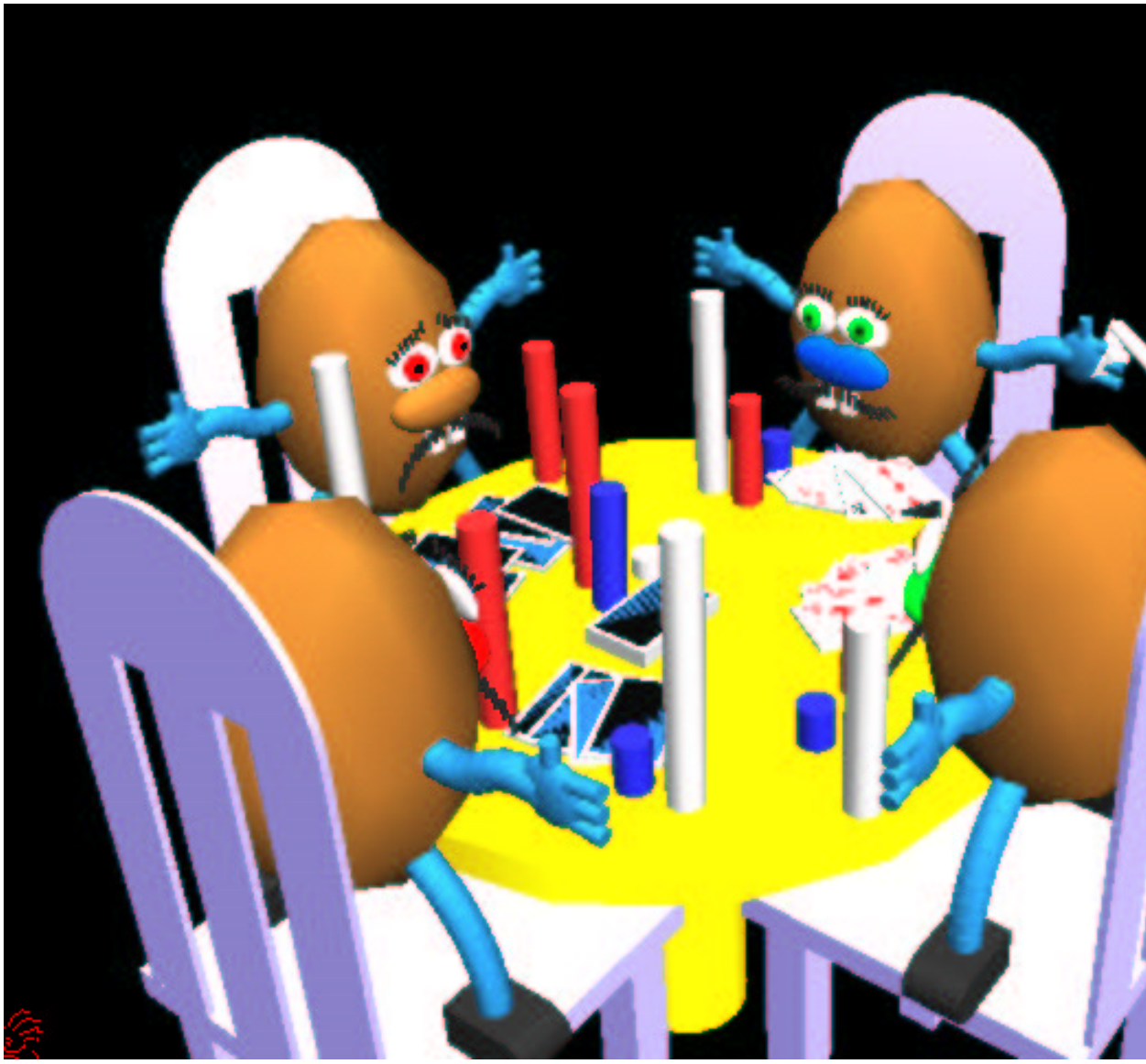
Juan Reyes (jcarlos)



Megan Galbraith (megan)

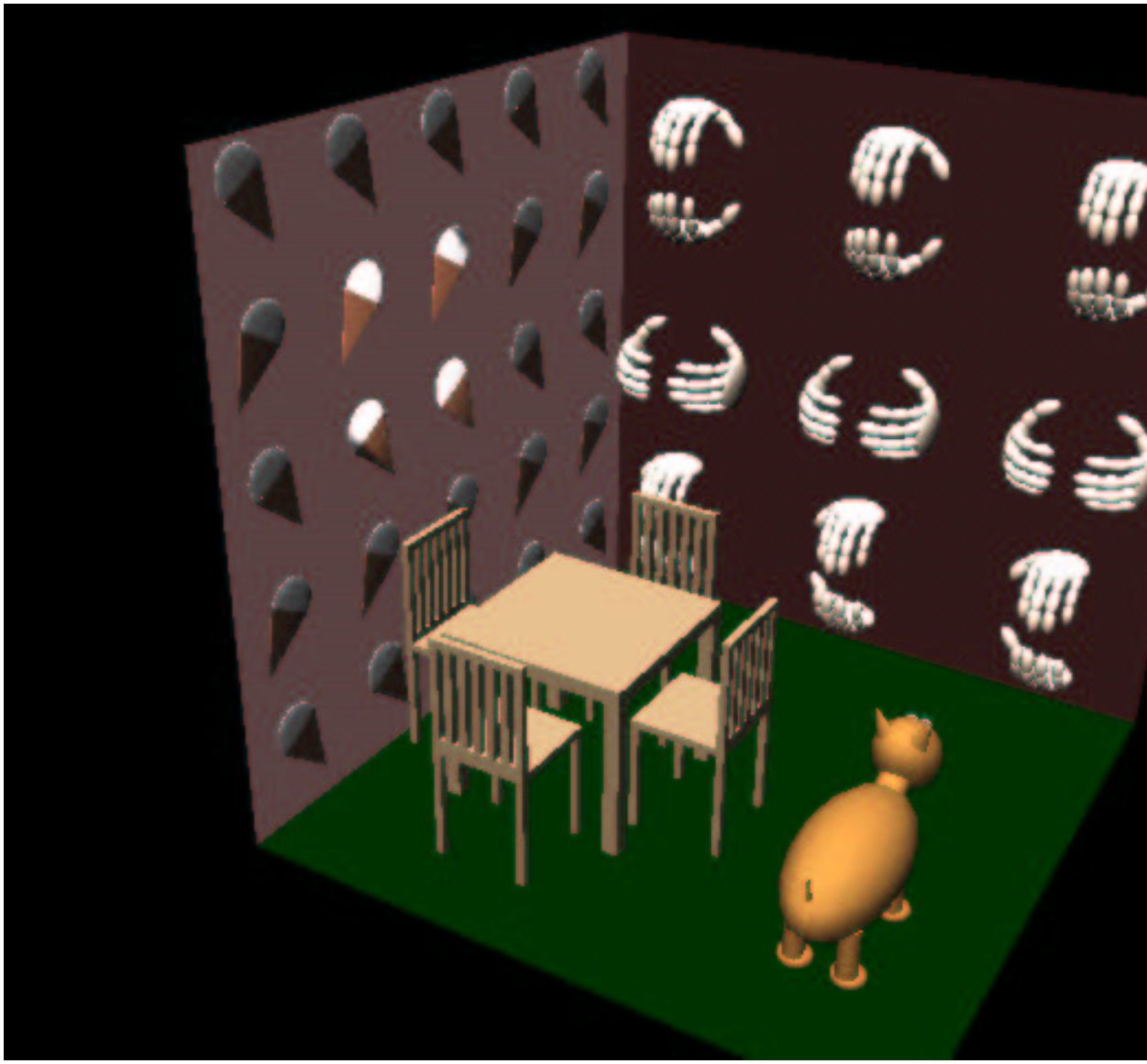


Meena Shah (mshah91)

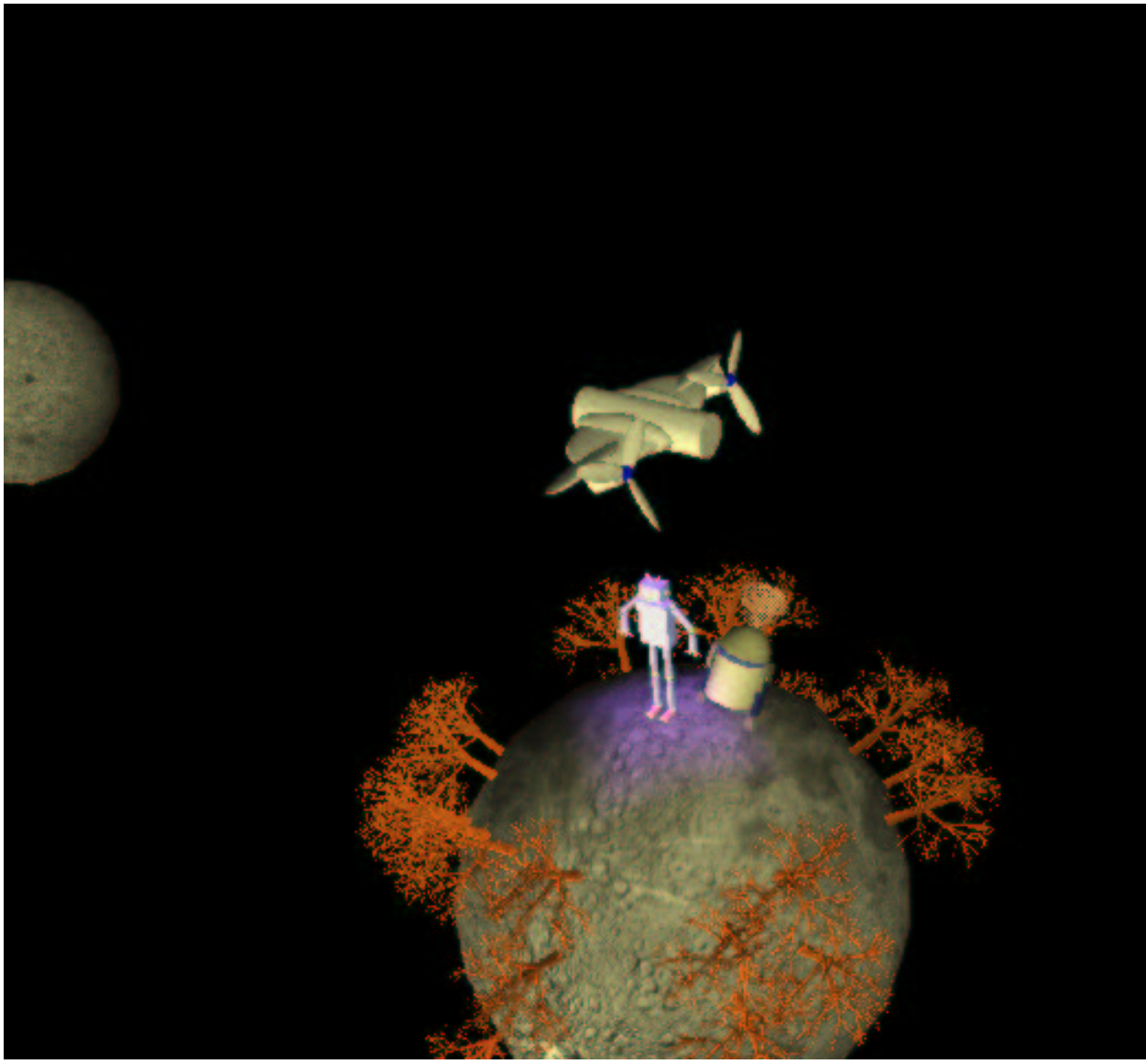


Naveen Goela (ngoela)

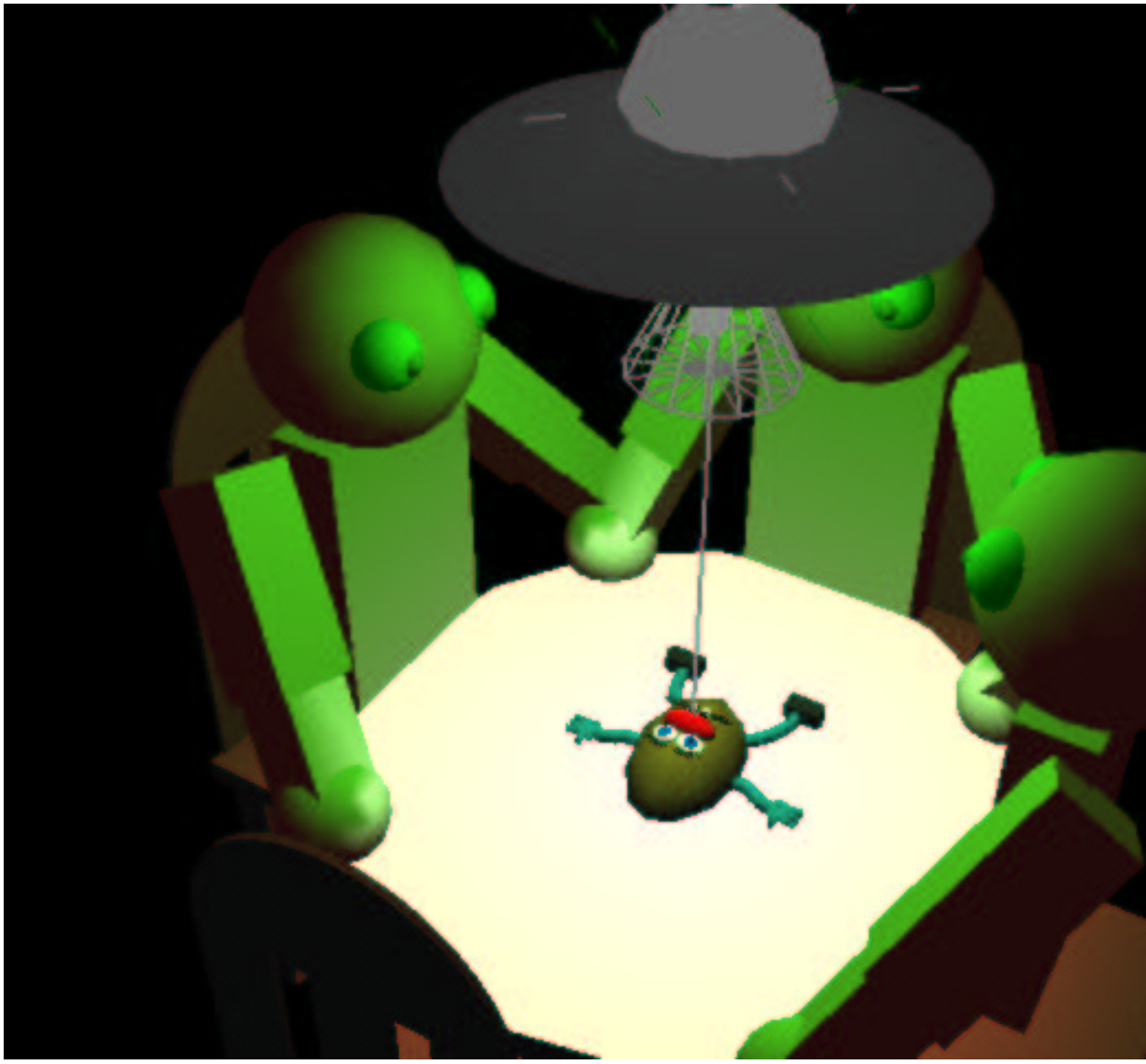
Philip Lee (philee)



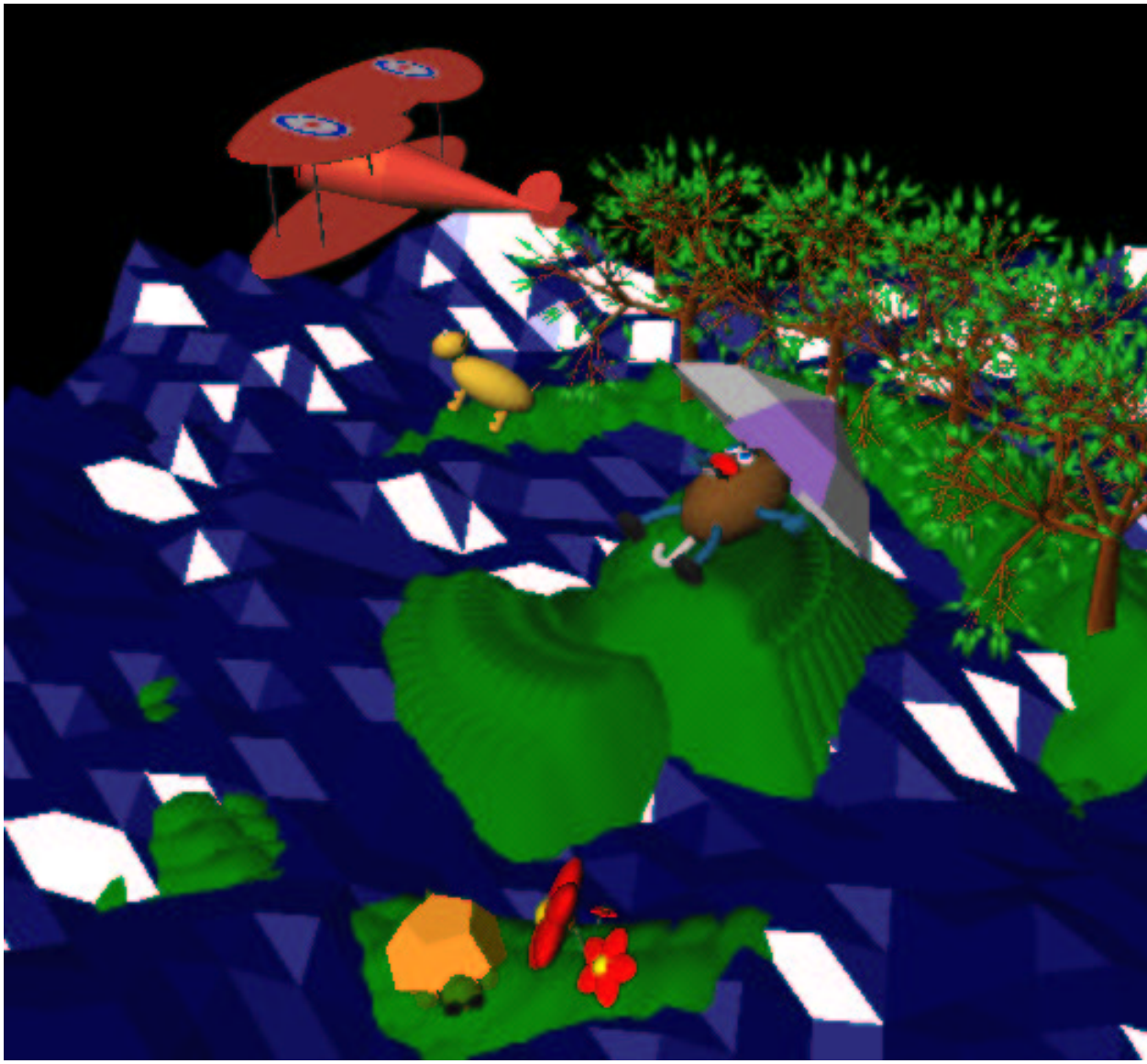
Pierre Poignant (poignant)



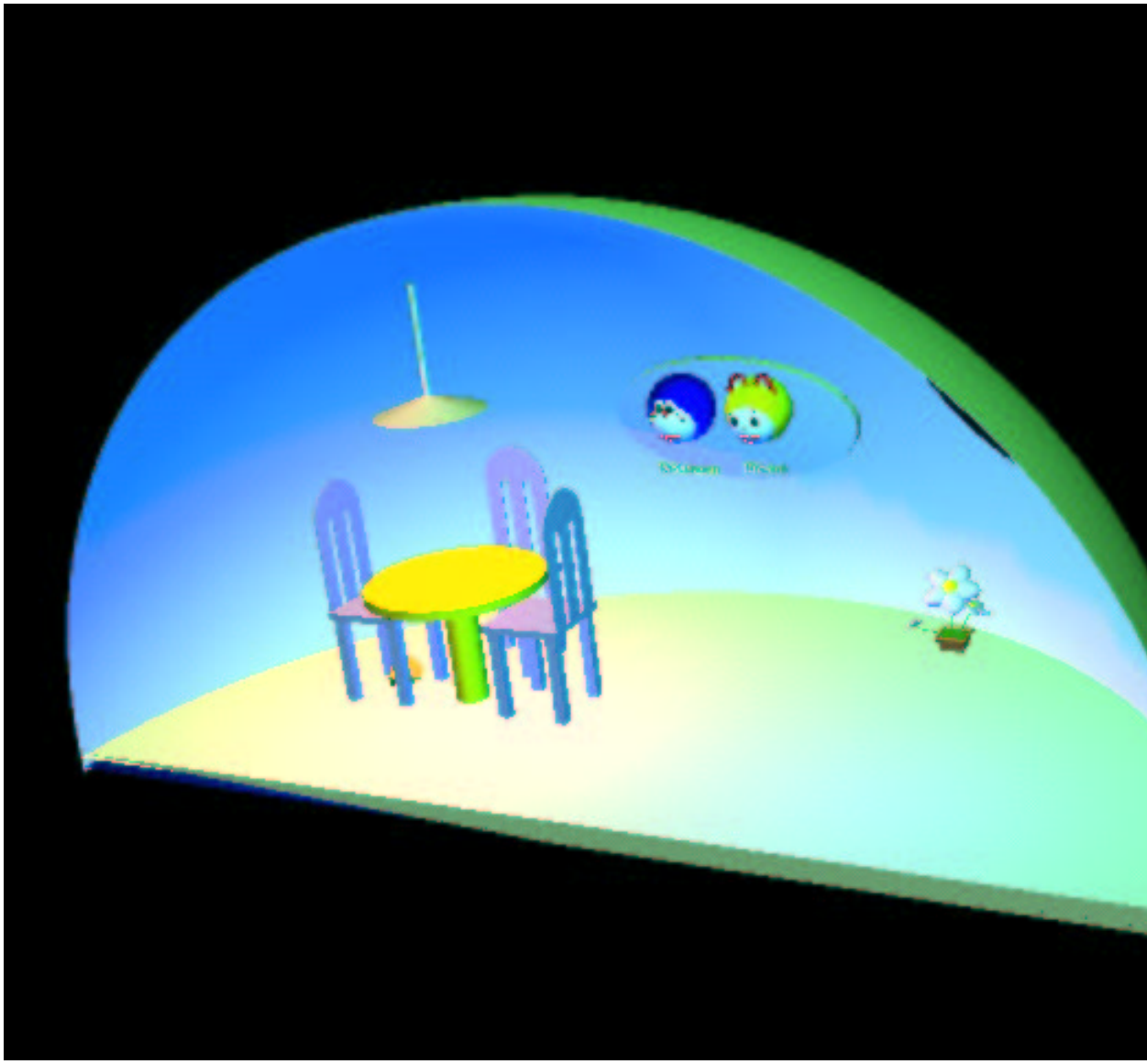
Conan Saunders (saunders)



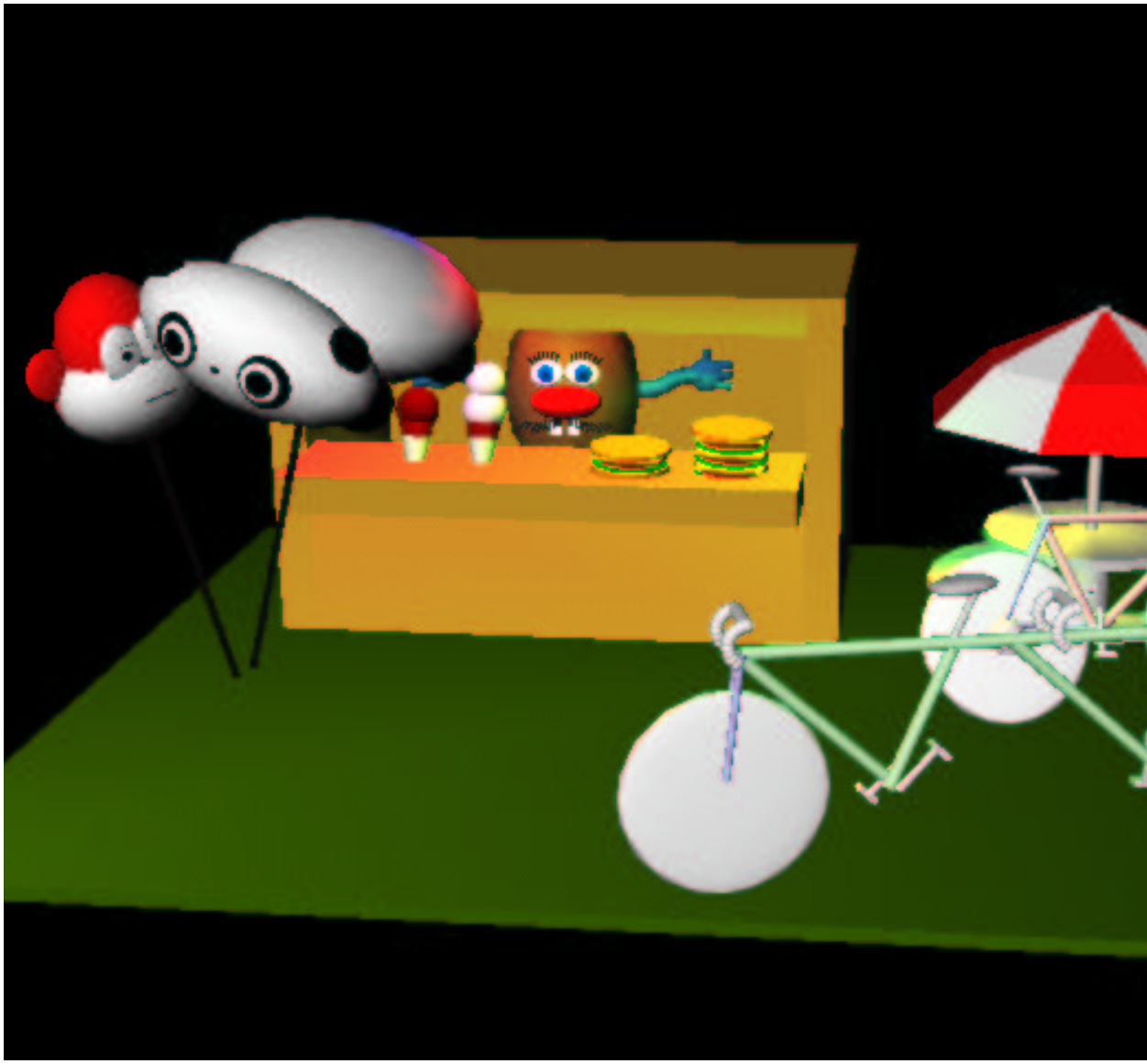
Xian Ke (xke)



Yao Li (yaol)

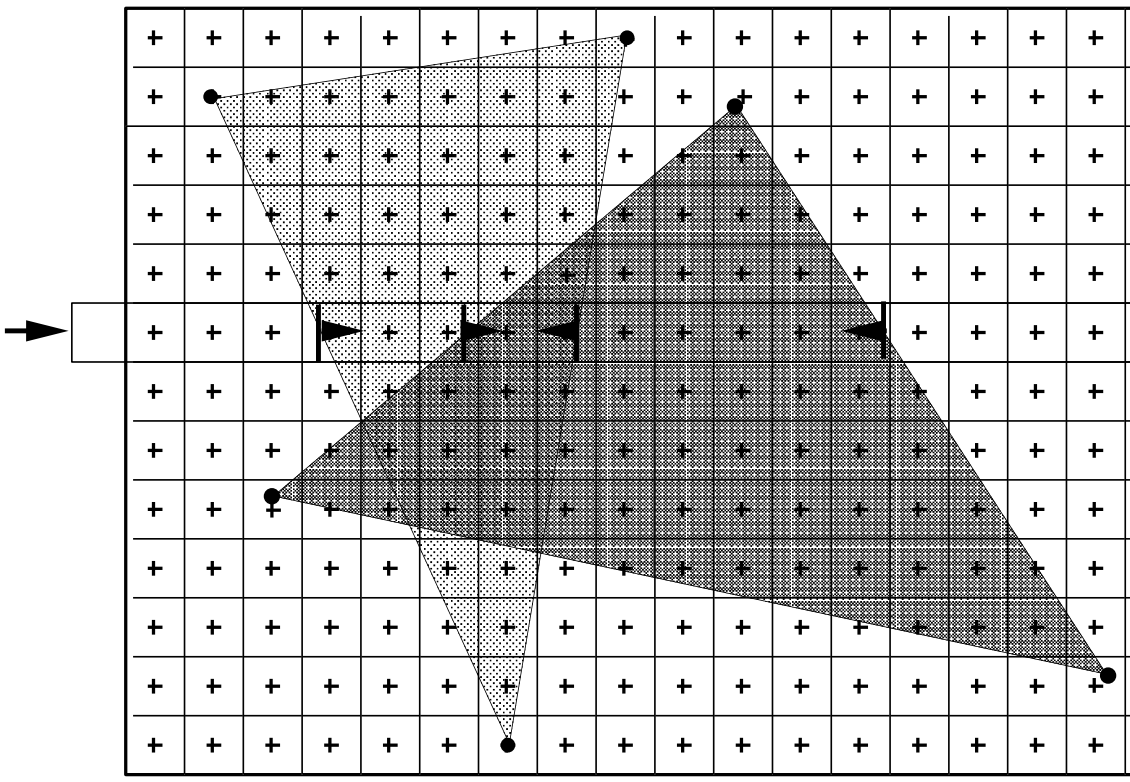


Esther Yoo (yoo)



Scan-line Hidden Surface Algorithms

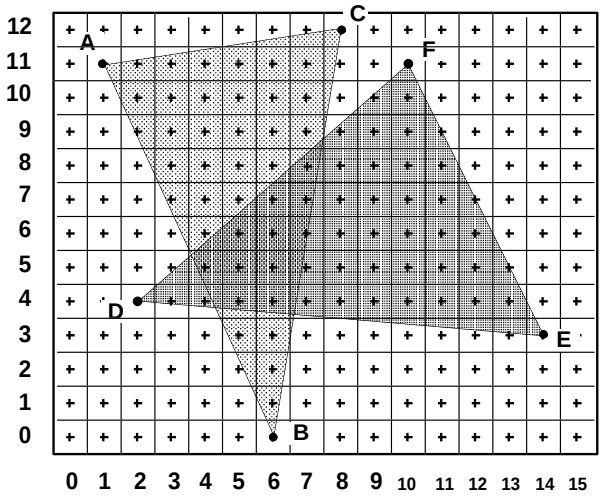
We'll work with just one *raster* of framebuffer memory (c
"Sweep" horizontal scanline upward over scene (from raste

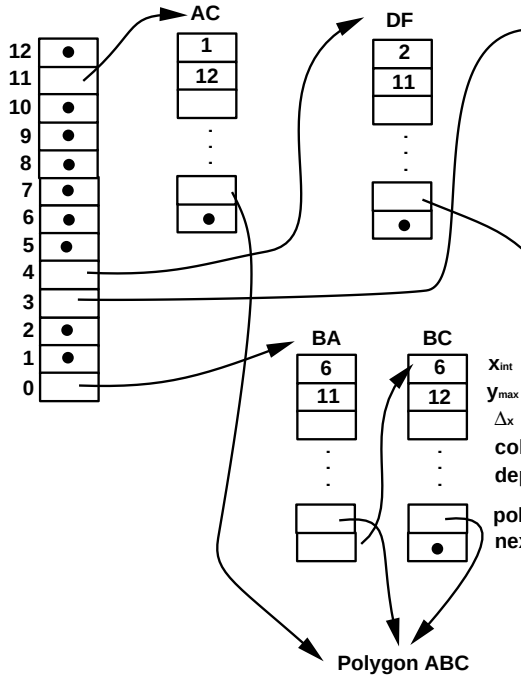


For now, assume polygons are **convex**, with no horizonta
  Exactly 0 or 2 edges at each scanline
  Each polygon has IN/OUT flag (used later)
Scan-line algorithm maintains two *invariants:*
  1. Data structure contains portion of scene intersected b
     current scanline, *ordered* by x intercept of edges
  2. Renderer outputs all pixels on current scanline before

Initialization

Precompute: Edge Table (ET), one entry per scan line





Each entry is a linked list of `EdgeRec`s, sorted by x_{int}, co

`float` y_{end}: y of top edge endpoint

`float` x_{int}, Δ_x: current x intersection, delta wrt y

`float` col_{curr}, Δ_{col}: current color, delta wrt y

`float` z_{curr}, Δ_z: current depth, delta wrt y

Pointer to polygon from which edge came (for matching

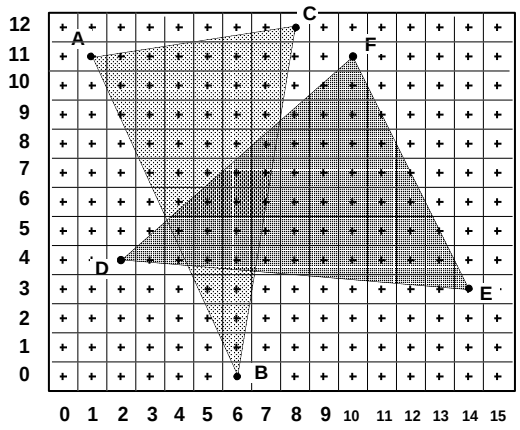
next: pointer to next record, or NULL

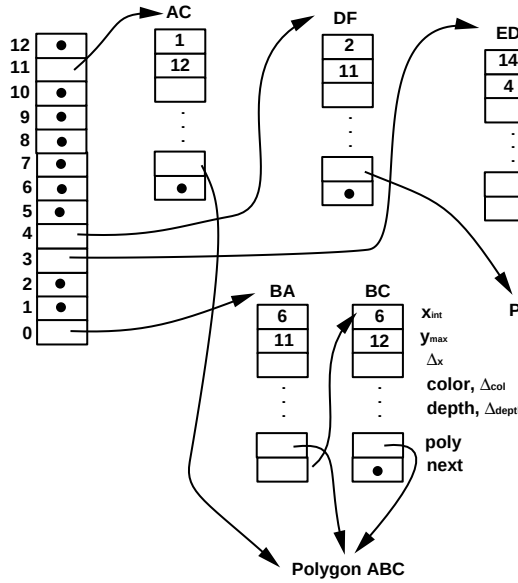
At start of frame, insert all edges into ET

... How long does precomputation take in total?

Initialization (cont.)

Input: list of polygons, each specified as ordered list of edg





Think of Edge Table as a bucketed list of **"events"**:

ET is static data structure

Event: start/finish of an edge (i.e., vertex)

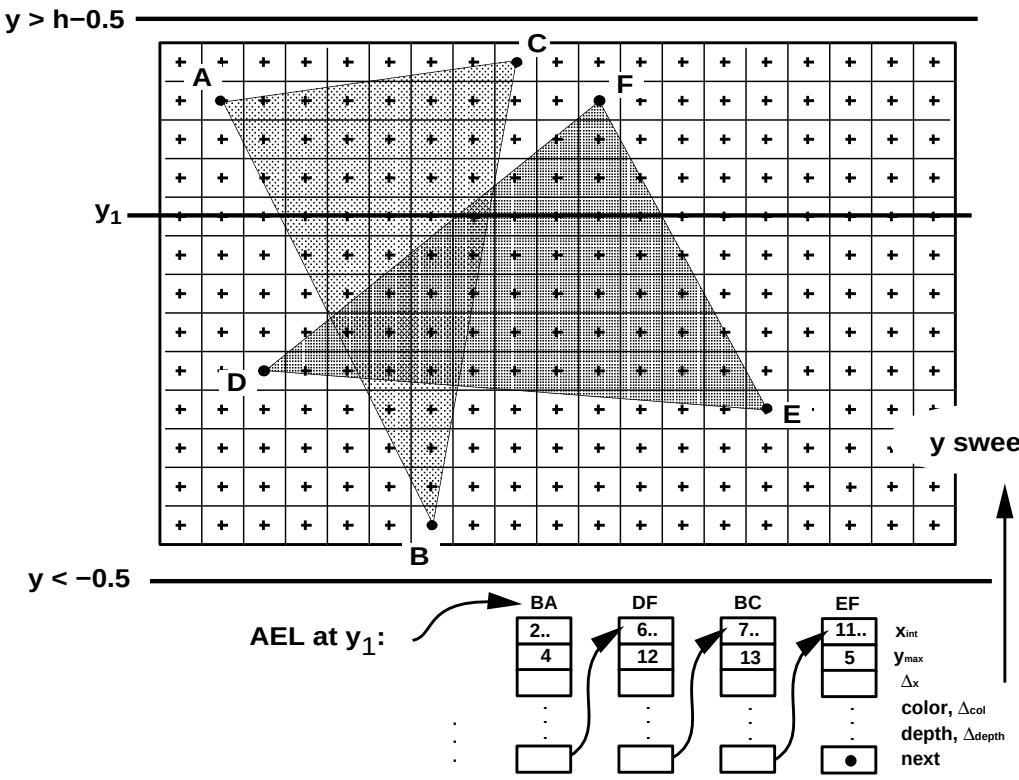
occurs within interval $[y, y+1)$

Within each bucket, events are sorted by x coordinate

Active Edge List

Active edge list (AEL) is a dynamic data structure

It is initially empty (scan line beneath bottom of viewpo



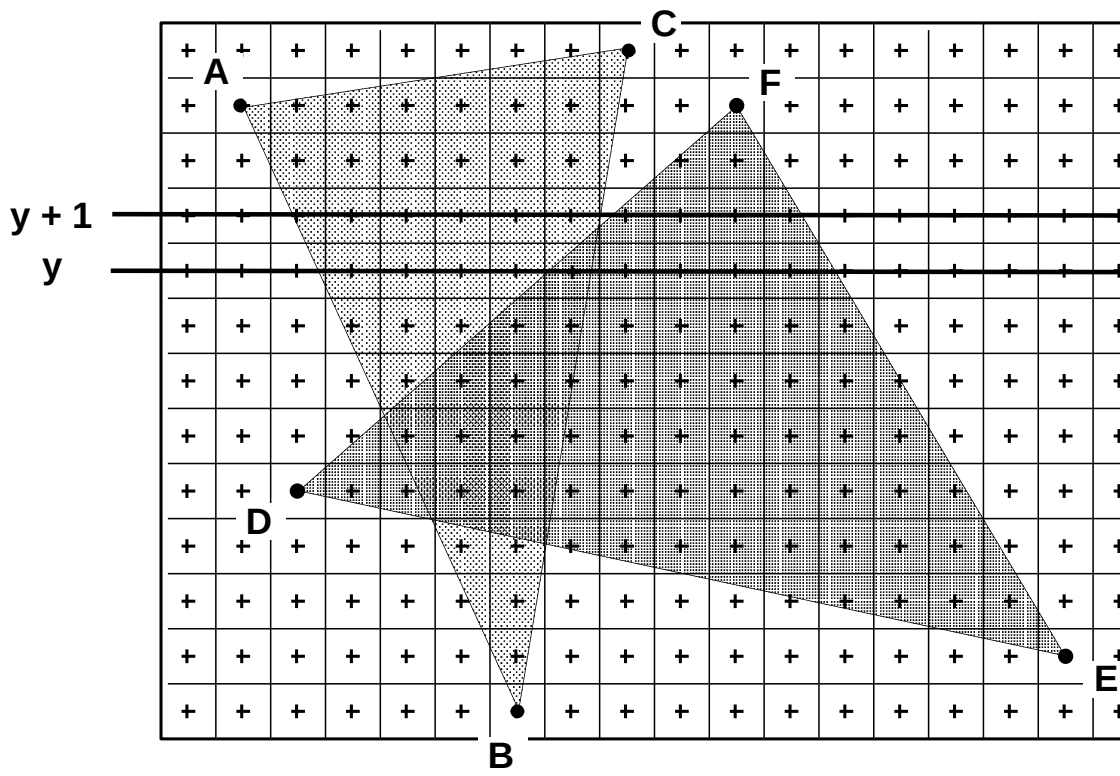
It is incrementally maintained to store

all edges intersecting scanline, ordered by x

It is empty at termination (scan line above top of viewp

(This is a useful sanity check for your implementation)

When Does AEL Change State?



When a vertex is encountered

I.e., when an edge begins or ends

All such events pre-stored in Edge Table!

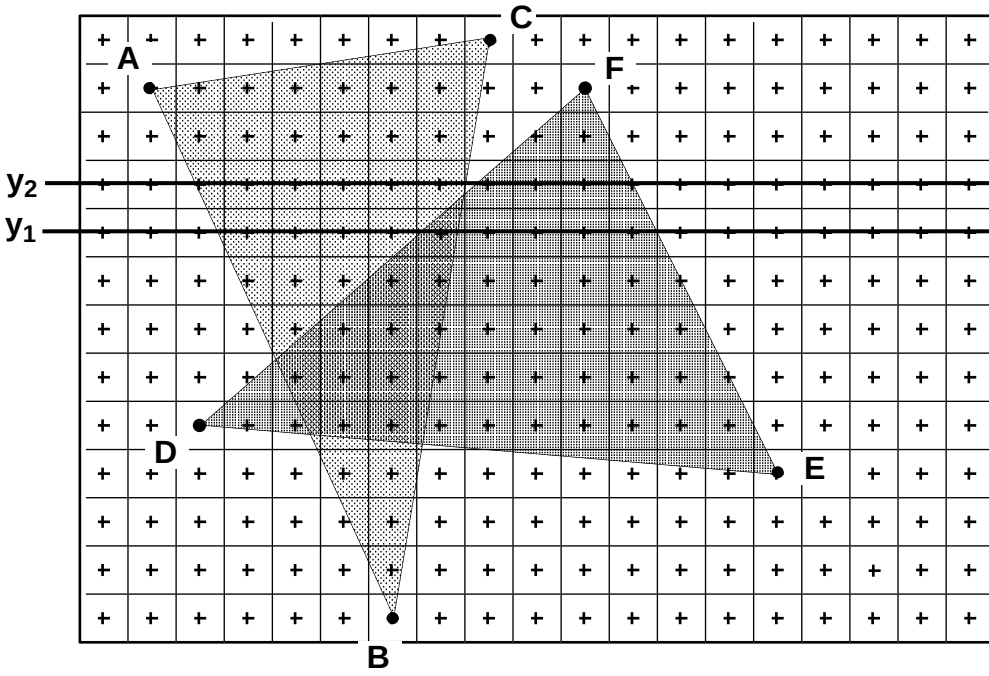
How do y pre-sorting, x pre-sorting improve efficiency?

When two edges change order along a scanline

I.e., when edges cross each other!

How to detect this efficiently?

Processing a Scanline: Span Rendering



Traverse AEL, left to right:

When edge encountered, toggle poly's IN/OUT flag

If entering polygon, scan for matching edge (how?)

Determine visible polygon *at each pixel center*

1) Evaluate z along each active span, & find min; or...

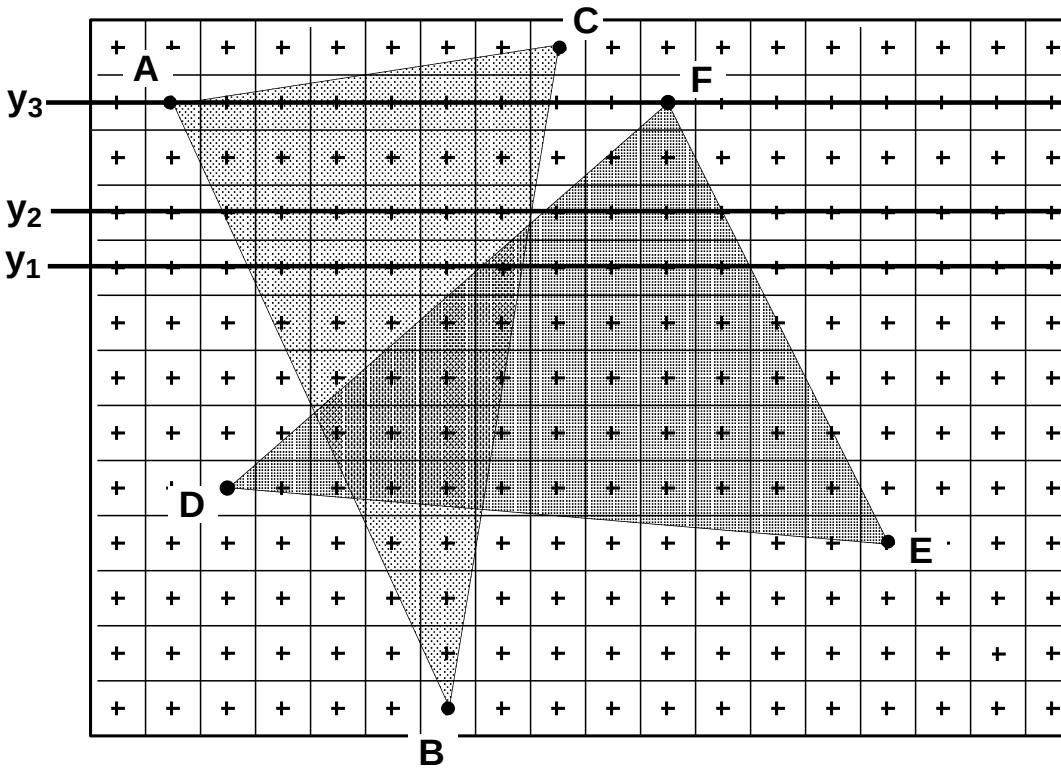
2) Render each span into one-raster z buffer; or...

3) Use more elegant ASL method (we'll see this in a m

Extract interpolated color from visible span

Use `setPixel(`x, C`)` to assign this pixel color to raster

Processing a Scanline: Maintenance



For each edge in AEL:

If edge ends in $(y-1, y]$, delete it from AEL

Otherwise, for next y: update the edge's x, color, z valu

Maintain AEL sorted by x (I.e., handle edge crossings. I

Increment scanline variable y

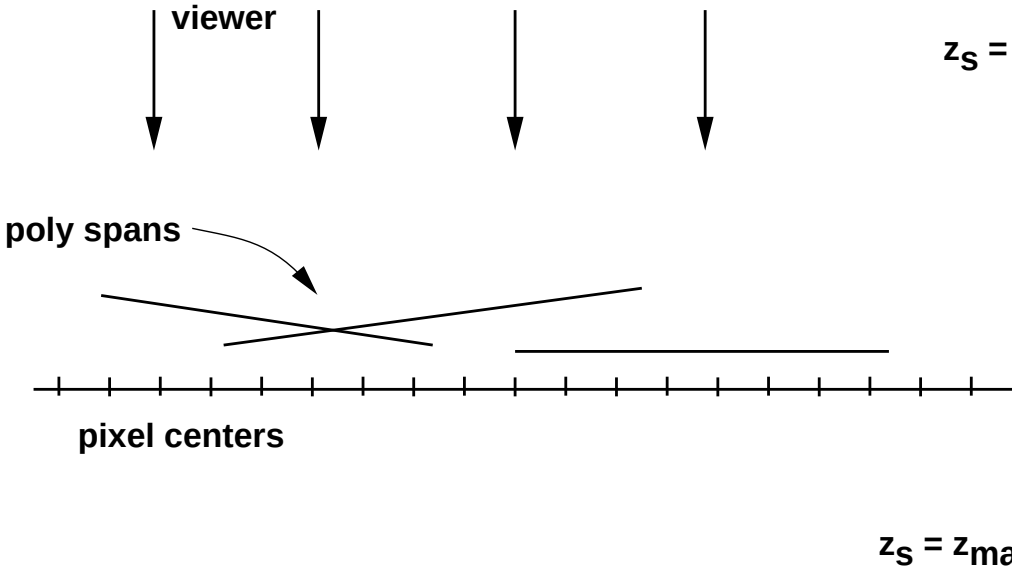
What is the spatial coherence here? How is it exploited fo

Question: suppose we don't have storage for even a *single*

Can we still generate a valid scan-converted image?

Raster Filling Without a Single Raster B

Use Active Span List (ASL)



Analogous to AEL ...

Except one dimension lower !

Sort span starts left to right; initialize color, depth, increm

Sweep across raster from $x = 0$ to $x = w - 1$

Track active spans in ASL

Maintain identity of span with minimum depth

Output pixel color for this span

Move to next x

If span complete, delete it;

Otherwise, increment its color and depth

Algorithm Summary:

Initialize Polygons, **ET, AEL**

  This takes $O(n \lg n)$ time worst case

For each scanline y

  Update AEL (insert edges from ET[y])

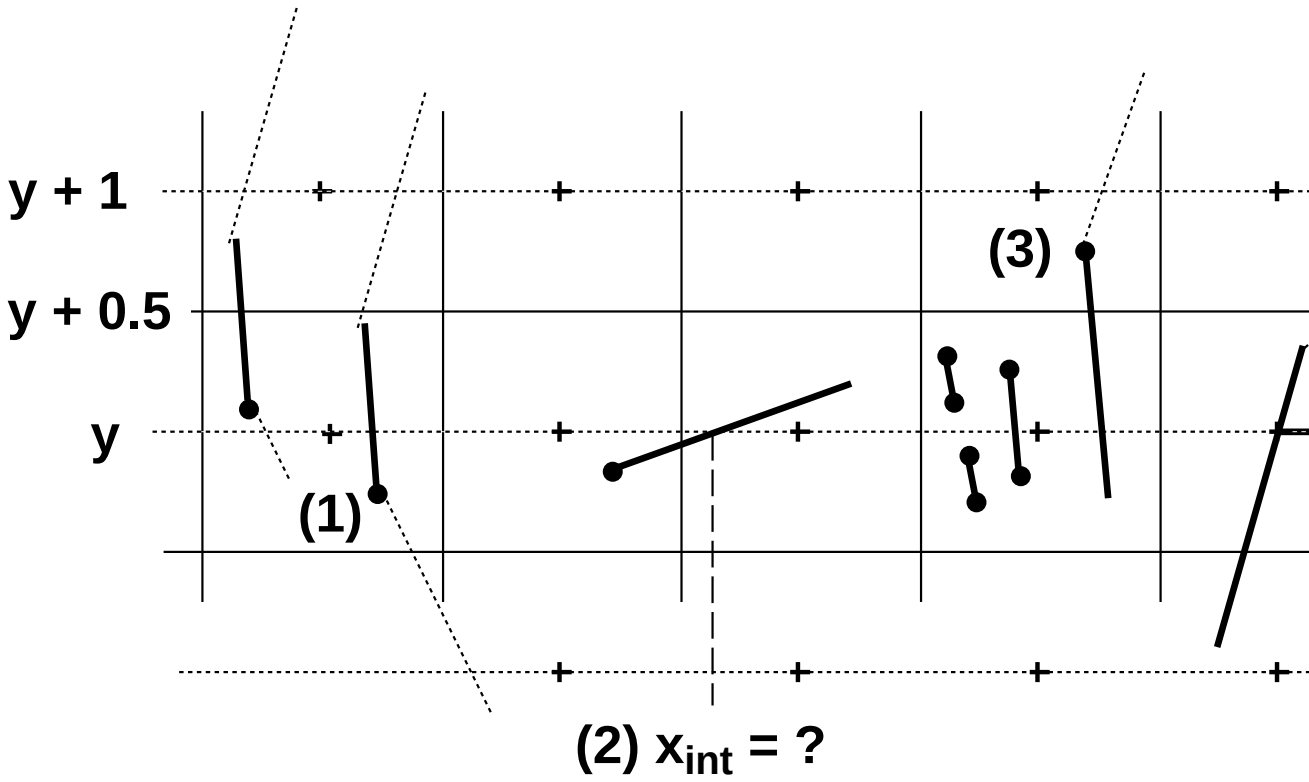
  Assign raster of pixels from AEL; output raster

  Update AEL (delete, increment, resort)

Clean up data structures

Scan Conversion Subtleties I

So far, example polygons have had vertices *integer* coordi
In practice (as in `ivscan`), vertices are floating-point valu



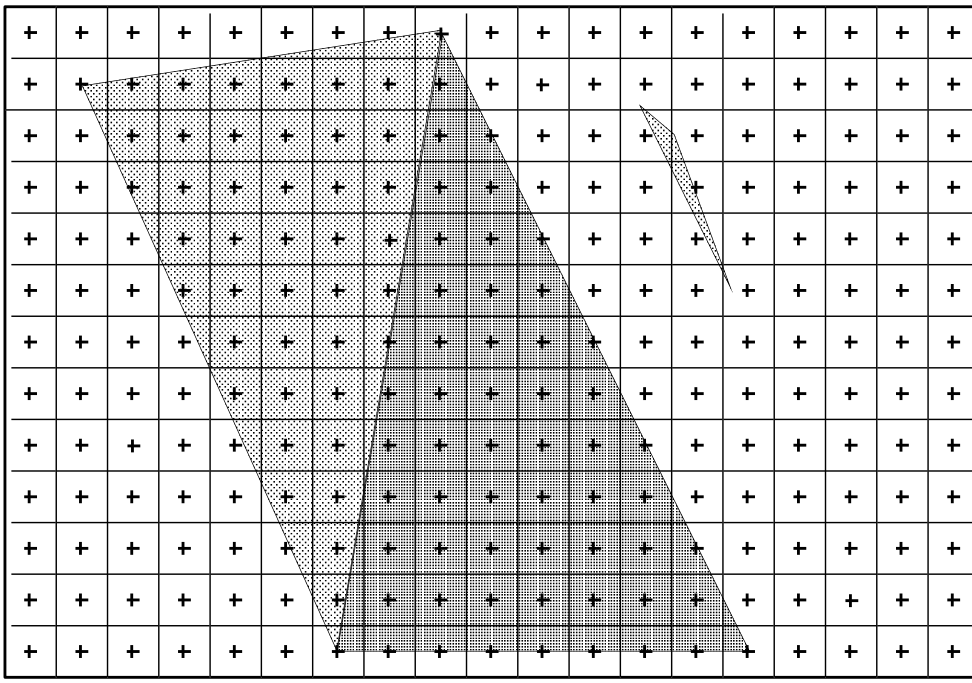
This introduces a number of subtle issues:

1. To which ET bucket does an edge beginning at $y = y$
2. What are the proper initial values for $x_{\text{int}}, \text{col}_{\text{curr}}, \text{z}_{\text{cu}}$
3. When should an edge ending at $y = y_{\max}$ be removed
4. Which pixels should be filled for a span from $x_{\text{int}1}..x_{\text{i}}$

Useful general principle: *think*, and formulate a simple, *co*
Helpful to draw (discrete) pixel coordinates and (idealized

Scan Conversion Subtleties II

Careful to handle *singularities* and *degeneracies*:



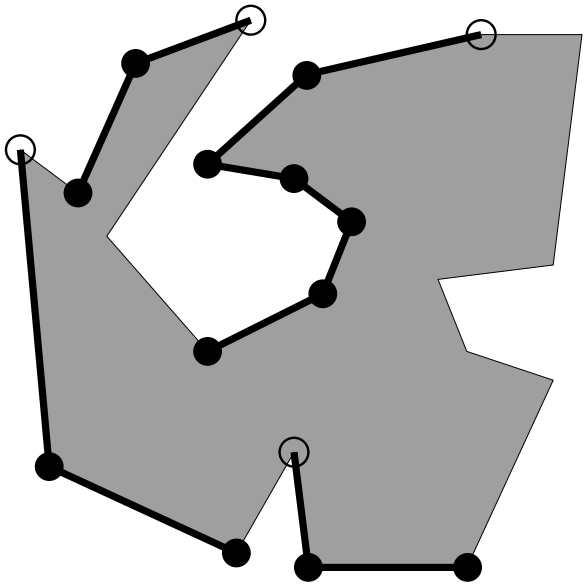
Omitted boundary pixels (causing gaps)
Twice-filled pixels (problem when blending, anti-aliasing
Sliver polygons (aliasing)
Horizontal edges (need consistent coverage rule)

Each polygon should assign or "own" certain pixels:
polygon owns all pixels centered in poly interior
polygon owns no pixels centered outside poly interior

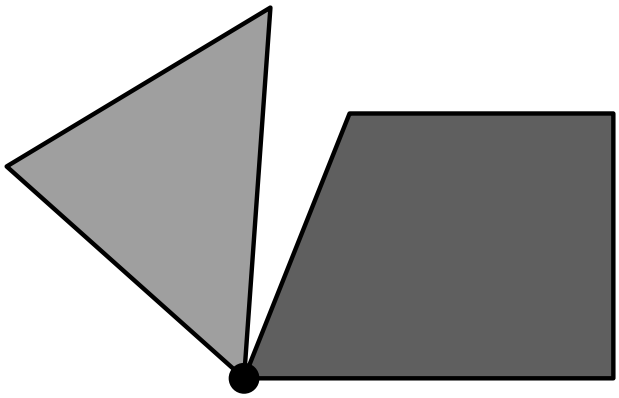
What about pixels on polygon boundary?
don't want pixels owned by multiple polygons
don't want to "drop" pixels (owned by no polygon)

Shadow Rule

"Shadow" Rule: polygon owns boundary pixel unless:
- Edge is horizontal, and on "top" of poly
- Edge is right-facing (normal has $n_x > 0$)
- Pixel at upper extremum of polygon

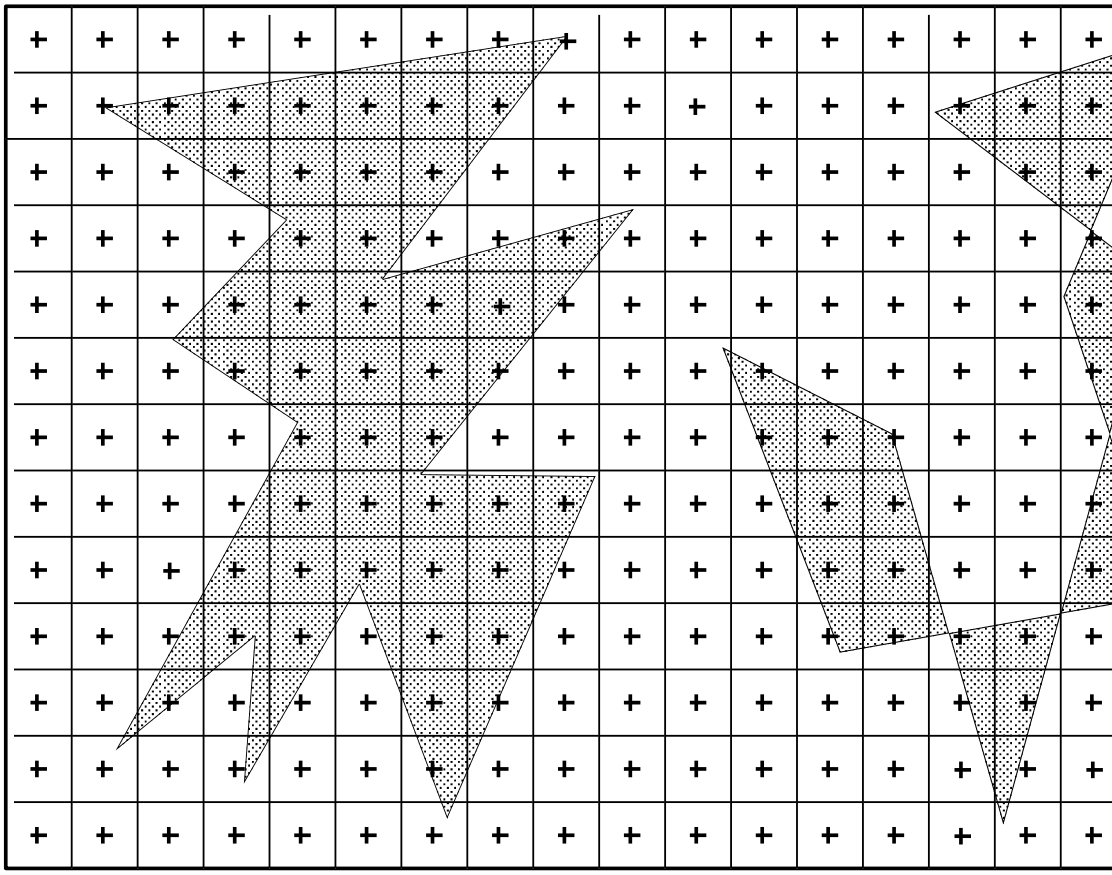


Still must handle special case of shared vertices!



Extensions

Non-convex polygons
Non-simple (i.e., self-intersecting) polygons



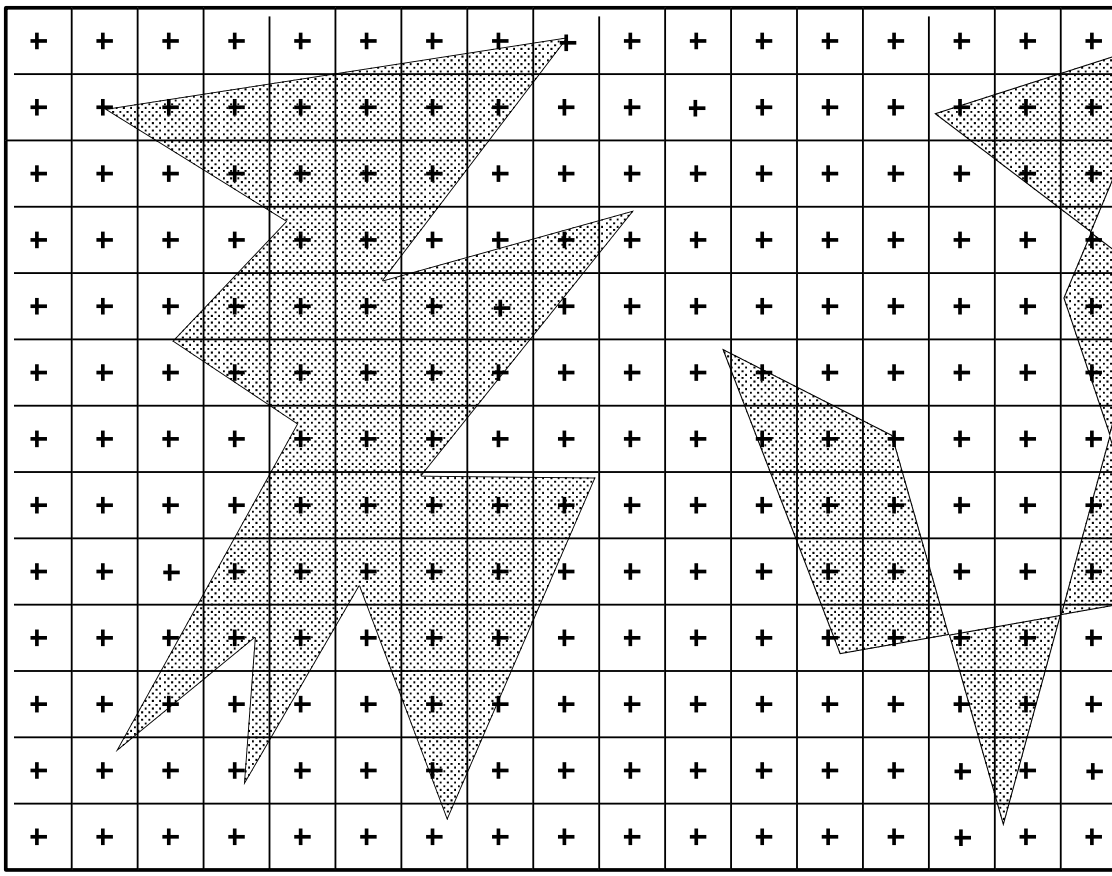
How do you handle these?

Extensions

Non-convex polygons

Non-simple (i.e., self-intersecting) polygons

- Maintain multiple active intervals per polygon
- Use polygon `in/out` flag defined earlier



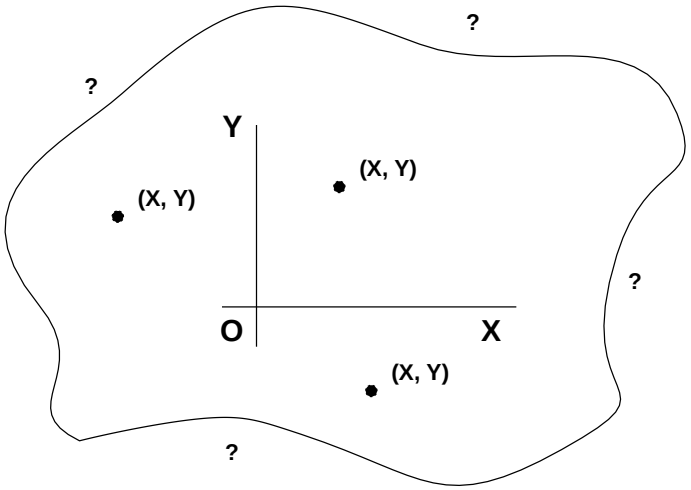
Spatial Coherence

A scene exhibits *spatial coherence* when:
  Visible surface identity, appearance change slowly across
Exploit spatial problem structure for efficiency
  Across scanlines: edge, polygon coherence
    Edge intersects scanline → likely that
      edge will intersect subsequent scanline
    ... Implication?
  Within scanline: span, depth coherence
    Span is visible at a pixel → likely
      that span is visible at adjacent pixels
    ... Implication?

What kind of scenes do not exhibit spatial coherence?
  ... Implications?

Homogeneous Coordinates Demystifie

Ordinary Cartesian coordinates (x, y) can represent only *j*



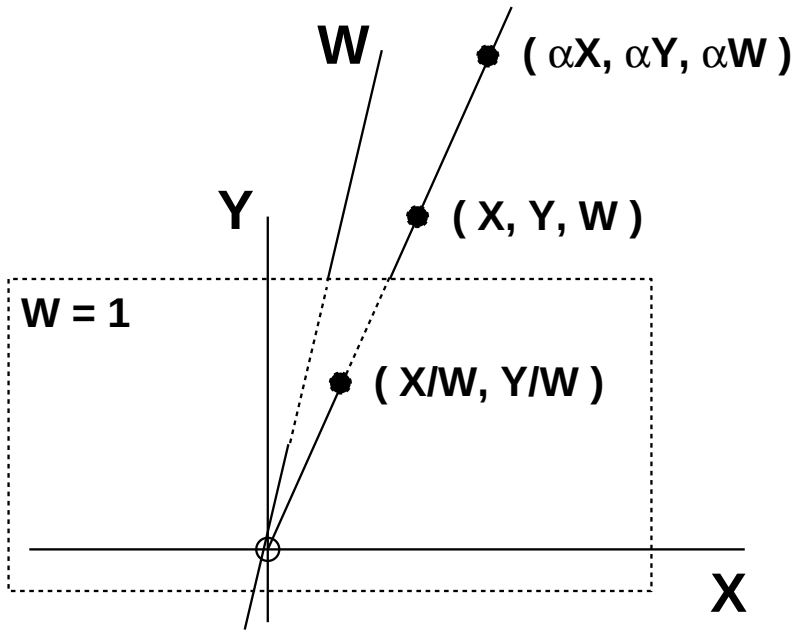
This excludes the set of points "at infinity" on the plane
We want to represent these "infinite points" with a *finite*
Moreover, translation inexpressible as 2×2 matrix:

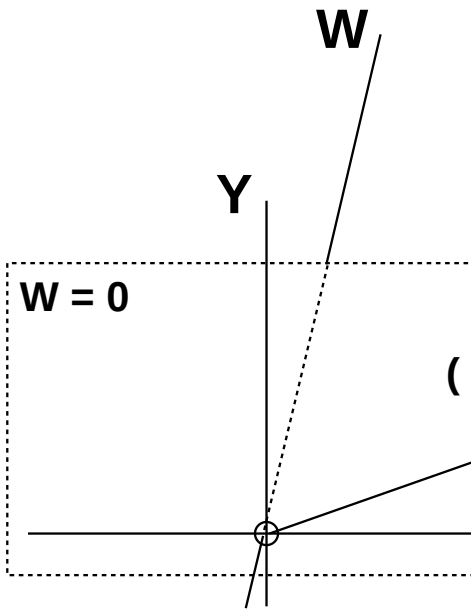
$$x' = x + t_x$$
$$y' = y + t_y$$
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} ? & ? \\ ? & ? \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

This makes composition of transformations awkward (no r
We want a coordinate representation that fixes both probl

2D Homogeneous Coordinates

Use representation (X, Y, W) for points





Divide by W, *truncate* to recover point $(\frac{X}{W}, \frac{Y}{W})$

Note that scalar multiplication has no effect:

$(X, Y, W) = (\alpha X, \alpha Y, \alpha W)$ for any $\alpha \neq 0$

This is why they are called "homogeneous coordinates"

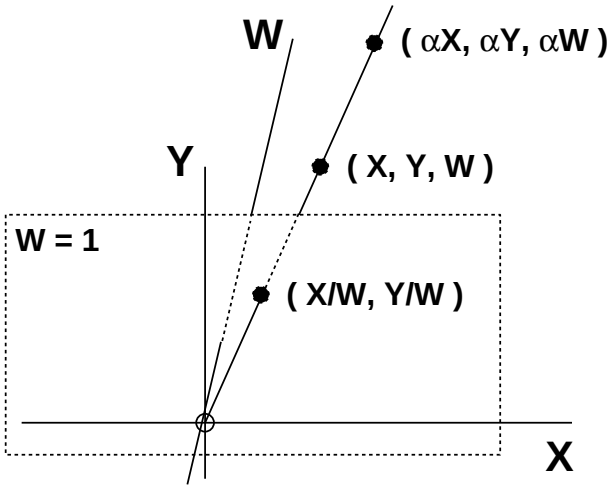
Projection operation maps X, Y, W coordinates with $W \neq$

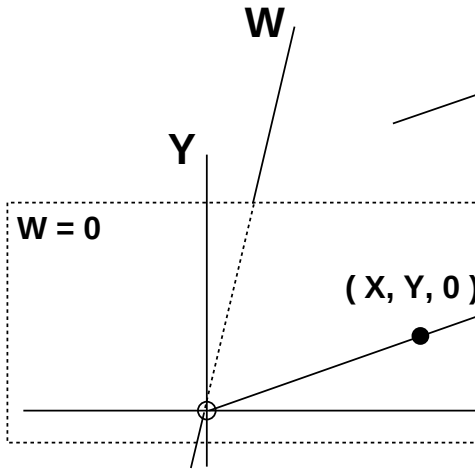
the projection plane $W = 1$ (copy of the Cartesian x, y

(Except for one X, Y, W point; which one?)

Ideal Points

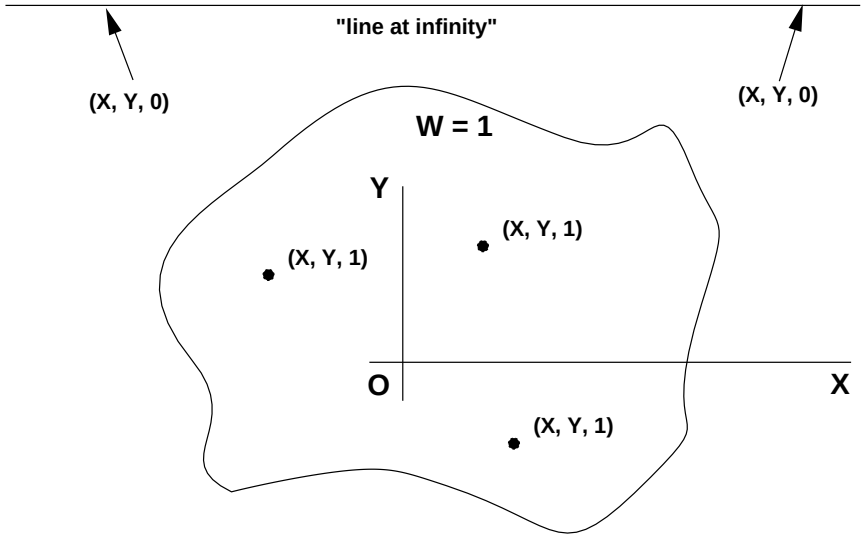
Consider the limit as $W \to 0$. What is this?





Also called an "ideal point" or "point at infinity"

Points with $W = 0$ called the "line at infinity"



Point, $(0, 0, 0)$ is excluded; it's not well-defined

Antipodal directions are *identical* in homogenous coordi

Cartesian plane, plus line at infinity comprise *Projective*

Advantages of 2D Homogeneous Coordin

Extension from 2-vector to 3-vector enables matrix formal

Example: *translation* as 3×3 matrix operation

$$
\begin{aligned}
x' &= x + t_x \\
y' &= y + t_y \\
\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} &= \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}
\end{aligned}
$$

And in combination with other transformations:

$$
\begin{aligned}
x' &= Ax + By + t_x \\
y' &= Cx + Dy + t_y \\
\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} &= \begin{pmatrix} A & B & t_x \\ C & D & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}
\end{aligned}
$$

(This makes possible ordinary composition through mult

Advantages of 2D Homogeneous Coordin

Unifies our representation of points, lines, and directions!

Points: $X, Y, 1$

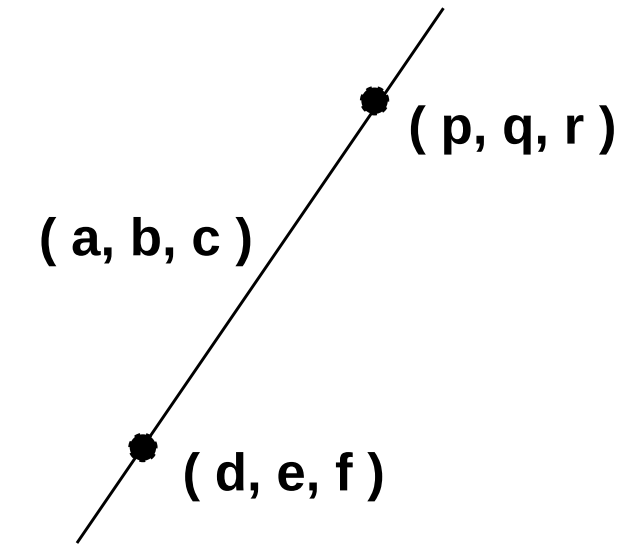
Lines: a, b, c such that $aX + bY + cW = 0$

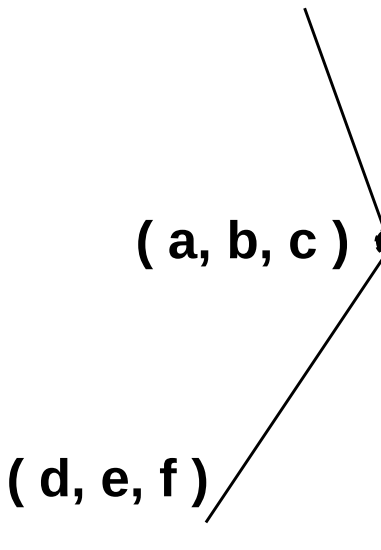
As inner product: $\begin{pmatrix} a & b & c \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ W \end{pmatrix} = 0$

Note that scaling coefficients a, b, c leaves line unchang

Directions: $X, Y, 0$

This allows us to generate true statements using *duality*,
simply by inter-changing the words "line" and "point" (





Example: "The line (a, b, c) is incident to two points $(d,$
... How do you compute the point lying on two lines?

Incidence of Lines and Points

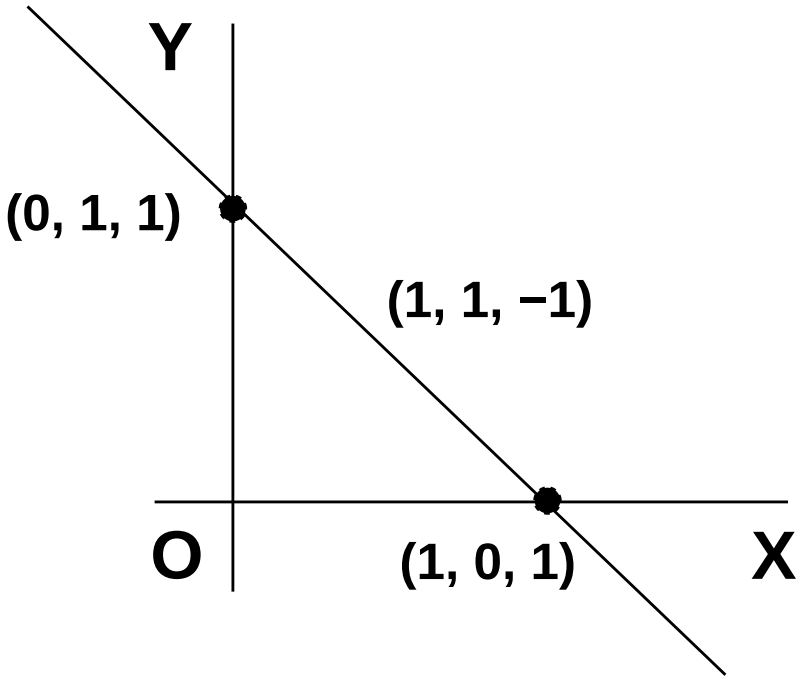
Using homogeneous coordinates, incidences arise from cros

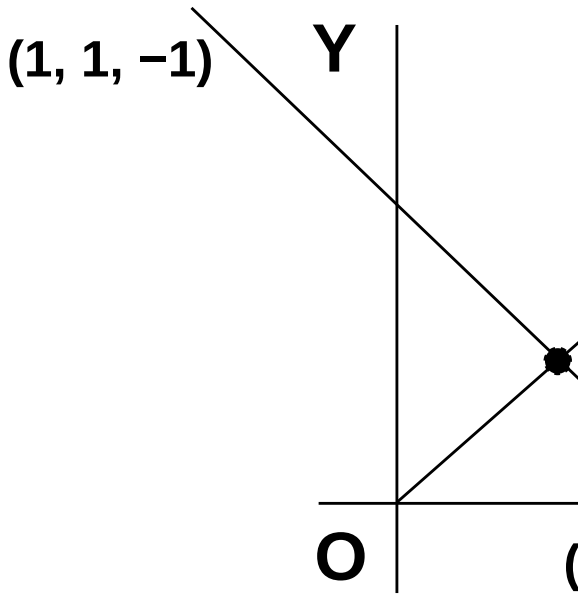
Two points make a line: $\mathbf{p} \otimes \mathbf{q} = \mathbf{l}$

Try it: points $(0, 1, 1)$ and $(1, 0, 1)$

Resulting line is $1x + 1y - 1 = 0$

What if points are coincident?





Duality: two lines make a point: $\mathbf{l} \otimes \mathbf{m} = \mathbf{p}$

Try it: lines $(1, 1, -1)$ and $(1, -1, 0)$

Resulting point is $(-1, -1, -2) = (\frac{-1}{-2}, \frac{-1}{-2}, \frac{-2}{-2})$

What if lines are identical? Parallel?

Example: lines $(1, 0, -1)$ and $(1, 0, 1)$ (Note result, ant

What are analogous operations in 3D?

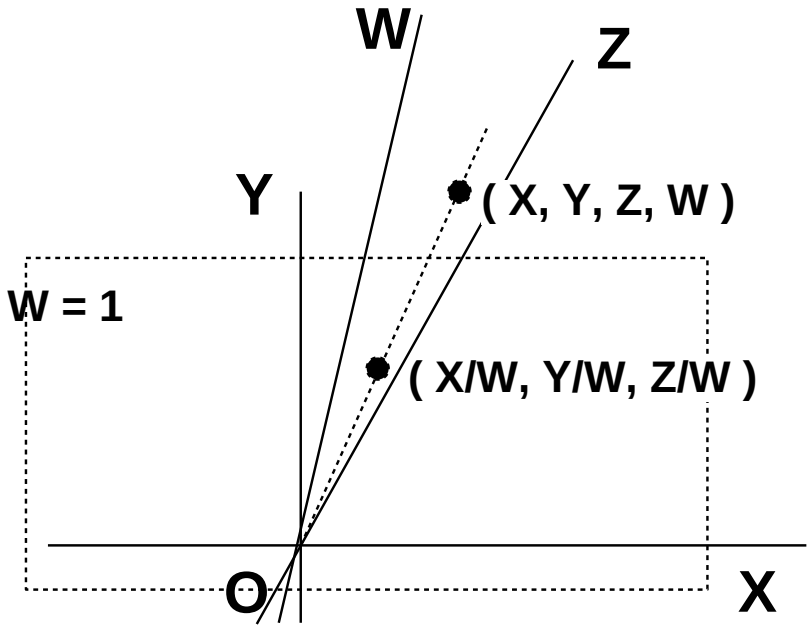
3D Homogeneous Coordinates

Use representation (X, Y, Z, W) for points

Divide by W, *drop* it to *recover* coordinates $(\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W})$

W "axis": no longer easily visualizable,

but there by analogy



Note: when $W = 0$, point at infinity $\equiv$ direction

Advantage of 3D Homogeneous Coordina

Formulate *translation* as 4×4 matrix operation
  (instead of as separate addition, as above)
Allow specification of "points at infinity": directions
  Infinitely distant point light sources
  Normals (specified by planes to which they are $\perp$)
Allows unified treatment of directions
  They behave as expected under matrix transformations
Formulate *projection* (non-linear) as matrix operation
  Map interior of view frustum to canonical parallelepiped

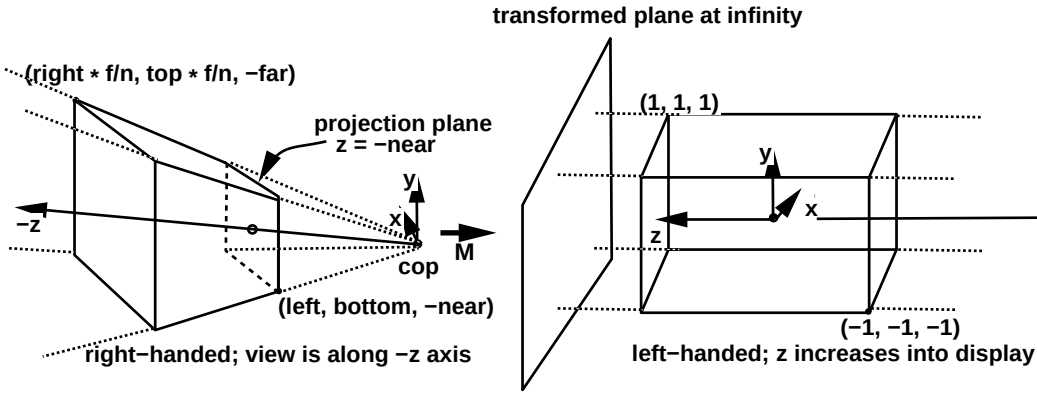
What happens to COP under perspectiv

Perspective matrix:

$$\mathbf{M} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Transform Center of Projection by $\mathbf{M}$:

$$\begin{aligned} \mathbf{COP'} &= \mathbf{M}\,\mathbf{COP} = \mathbf{M} \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & -\frac{2fn}{f-n} & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & -1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} \end{aligned}$$



Transforming Plane Equations

Plane equation $Ax + By + Cz + D = 0$ written in homog

$$\begin{pmatrix} A & B & C & D \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = 0$$

Suppose space is transformed by transformation $\mathbf{M}$
What "happens" to plane equation $\mathbf{H}$ under $\mathbf{M}$?

$$\mathbf{Hp} = 0$$

For any (non-singular) transformation matrix $\mathbf{M}$

$$\mathbf{HM}^{-1}\mathbf{Mp} = 0$$

Which can be rewritten (reparenthesized) as:

$$(\mathbf{HM}^{-1})(\mathbf{Mp}) = 0$$

Thus the transformed plane equation **H'** is

$$\mathbf{HM}^{-1}$$

Plane $\mathbf{H}$, under action of $\mathbf{M}$, becomes $\mathbf{HM}^{-1}$

What happens to plane through COP?

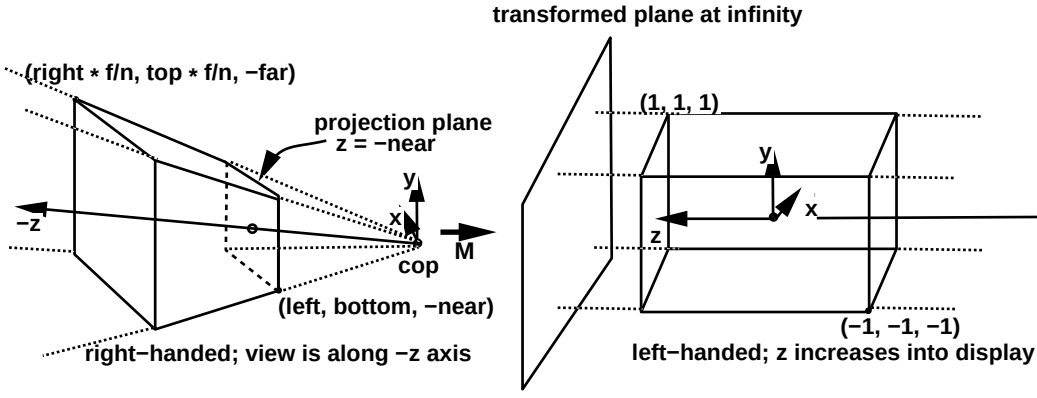
Plane postmultiplied by matrix *inverse*:

$$\mathbf{M}^{-1} = \begin{pmatrix} \frac{r-l}{2n} & 0 & 0 & \frac{r+l}{2n} \\ 0 & \frac{t-b}{2n} & 0 & \frac{t+b}{2n} \\ 0 & 0 & 0 & -1 \\ 0 & 0 & \frac{1}{2f}-\frac{1}{2n} & \frac{1}{2f}+\frac{1}{2n} \end{pmatrix}$$

consider plane $z = 0$, $\mathbf{H} = \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix}$

$$\begin{aligned} \mathbf{H}' = \mathbf{H}\mathbf{M}^{-1} &= \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} \mathbf{M}^{-1} \\ &= \begin{pmatrix} 0 & 0 & 0 & -1 \end{pmatrix} \end{aligned}$$

What points lie on $\mathbf{H}'$?



What happens to projection plane?

Again, postmultiply by matrix inverse:

$$\mathbf{M} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

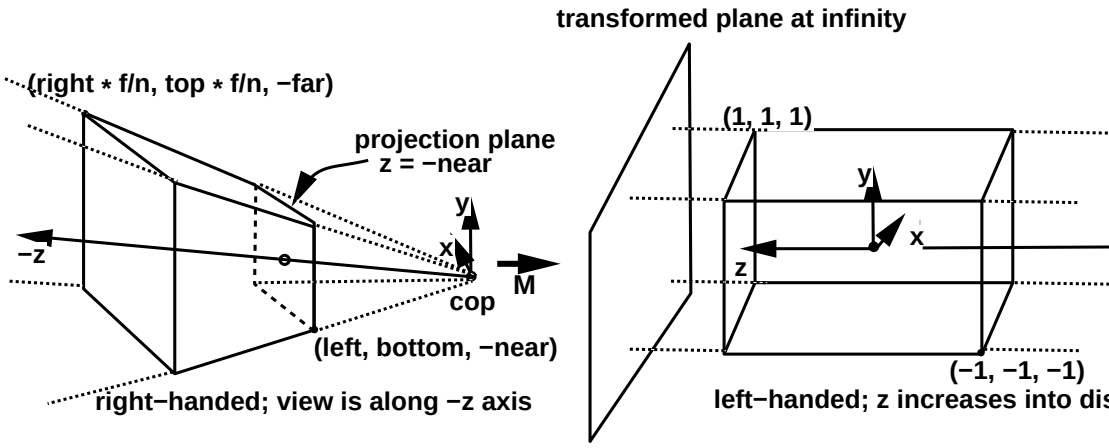
$$\mathbf{M}^{-1} = \begin{pmatrix} \frac{r-l}{2n} & 0 & 0 & \frac{r+l}{2n} \\ 0 & \frac{t-b}{2n} & 0 & \frac{t+b}{2n} \\ 0 & 0 & 0 & -1 \\ 0 & 0 & \frac{1}{2f} - \frac{1}{2n} & \frac{1}{2f} + \frac{1}{2n} \end{pmatrix}$$

consider plane $z = -n$, $\mathbf{H} = \begin{pmatrix} 0 & 0 & 1 & n \end{pmatrix}$

$$\begin{aligned} \mathbf{H}' = \mathbf{H}\mathbf{M}^{-1} &= \begin{pmatrix} 0 & 0 & 1 & n \end{pmatrix} \mathbf{M}^{-1} \\ &= \begin{pmatrix} 0 & 0 & \frac{n-f}{2f} & \frac{n-f}{2f} \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 1 & 1 \end{pmatrix} \end{aligned}$$

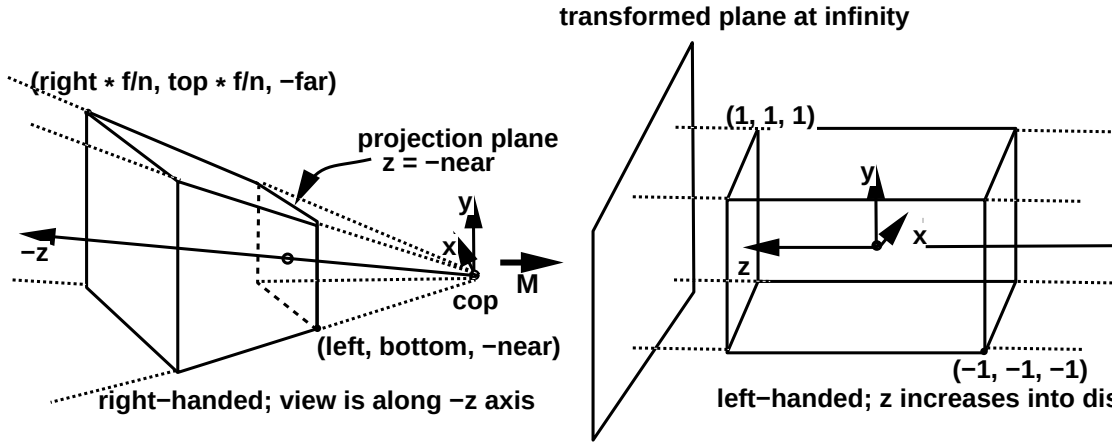
Transformation of projection plane (con

But this is the plane $z + 1 = 0$, or $z = -1$!



What happens to plane at infinity?

Its equation is $\mathbf{H} = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$



$$\begin{aligned} \mathbf{H}' = \mathbf{H}\mathbf{M}^{-1} &= \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{M}^{-1} \\ &= \begin{pmatrix} 0 & 0 & n-f & n+f \end{pmatrix} \end{aligned}$$

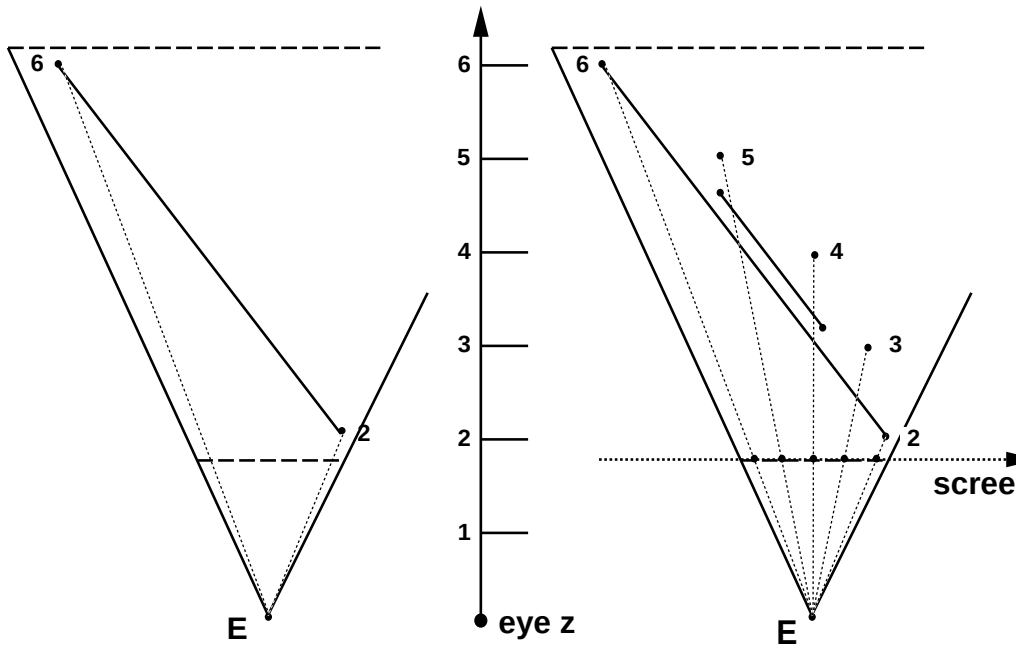
Suppose $n = 1,\ f = 2$. Then

$$\mathbf{H}' = \begin{pmatrix} 0 & 0 & -1 & 3 \end{pmatrix}$$

This is the plane $z = \frac{-(n+f)}{n-f} = 3$!

Screen Space Depth Interpolation

Pitfall of naive (eye-z) depth interpolation:



This is what happens when you interpolate

Eye z in **screen space**

(Geometric equivalent of Gouraud pitfall covered earlier)

How to interpolate depth correctly? Hints:

Work through homogeneous computation of z

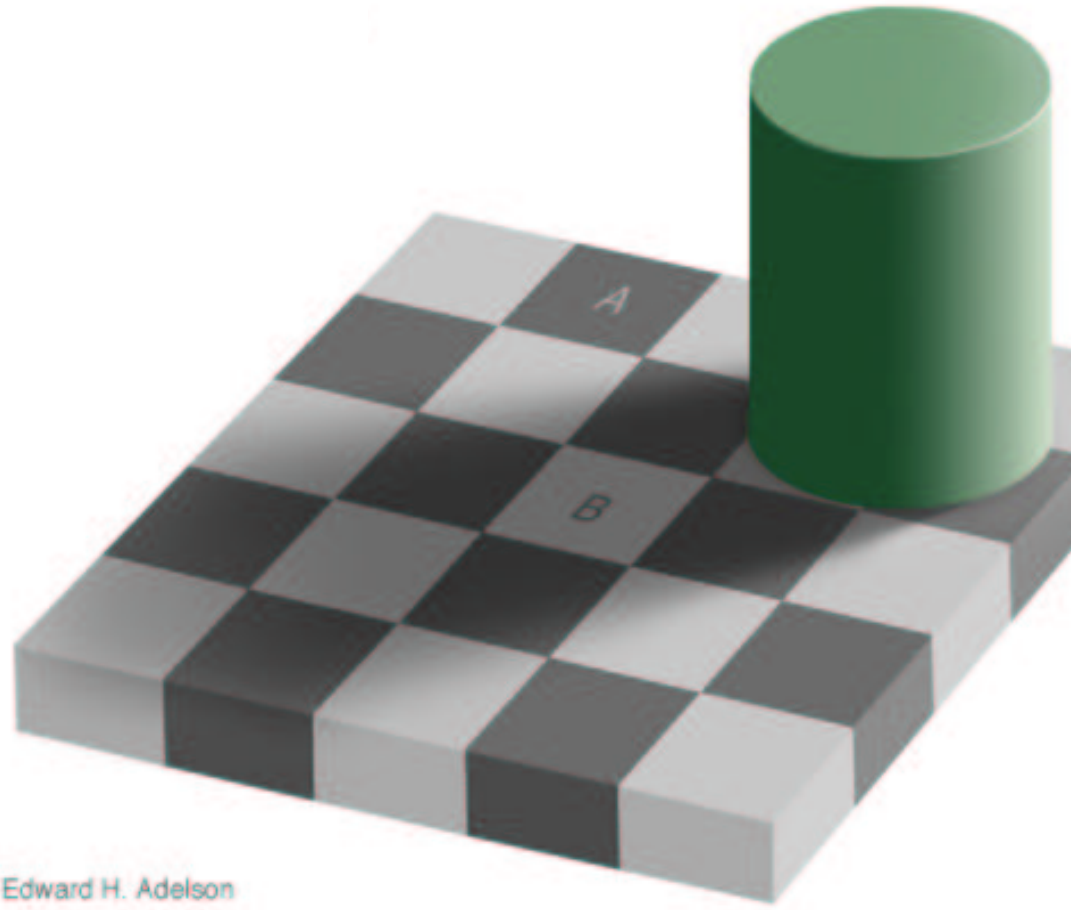
Read through comments in `ivscan` source:

`EdgeRec.h`

`ScanWrap.C`

Lighting Perception

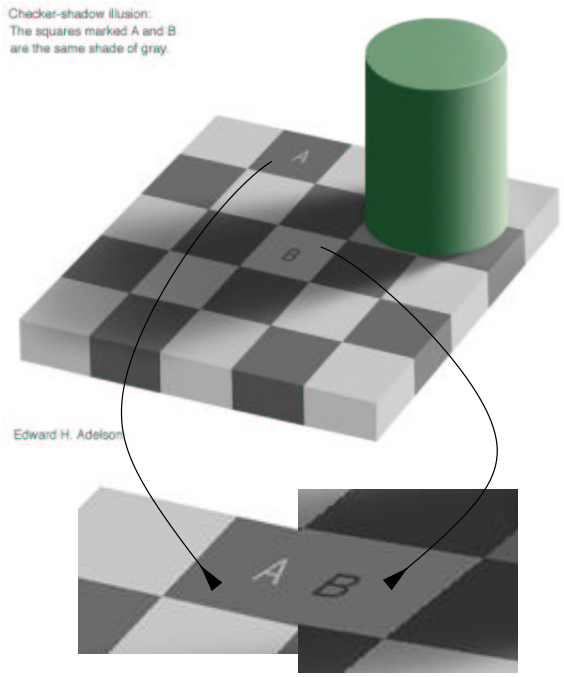
Image designed by Prof. Ted Adelson of BCS:



Which is darker, check A or B ?

Lighting Perception

They are the same! Why?



Visual system tries to determine intrinsic color of every ob

Local contrast (difference from "surround")

Check B surrounded by darker checks → looks light

Check A surrounded by lighter checks → looks dark

Shadow boundaries vs. texture boundaries

Texture (reflectance, color) boundaries usually sharp

Shadow boundaries usually fuzzy (why?)

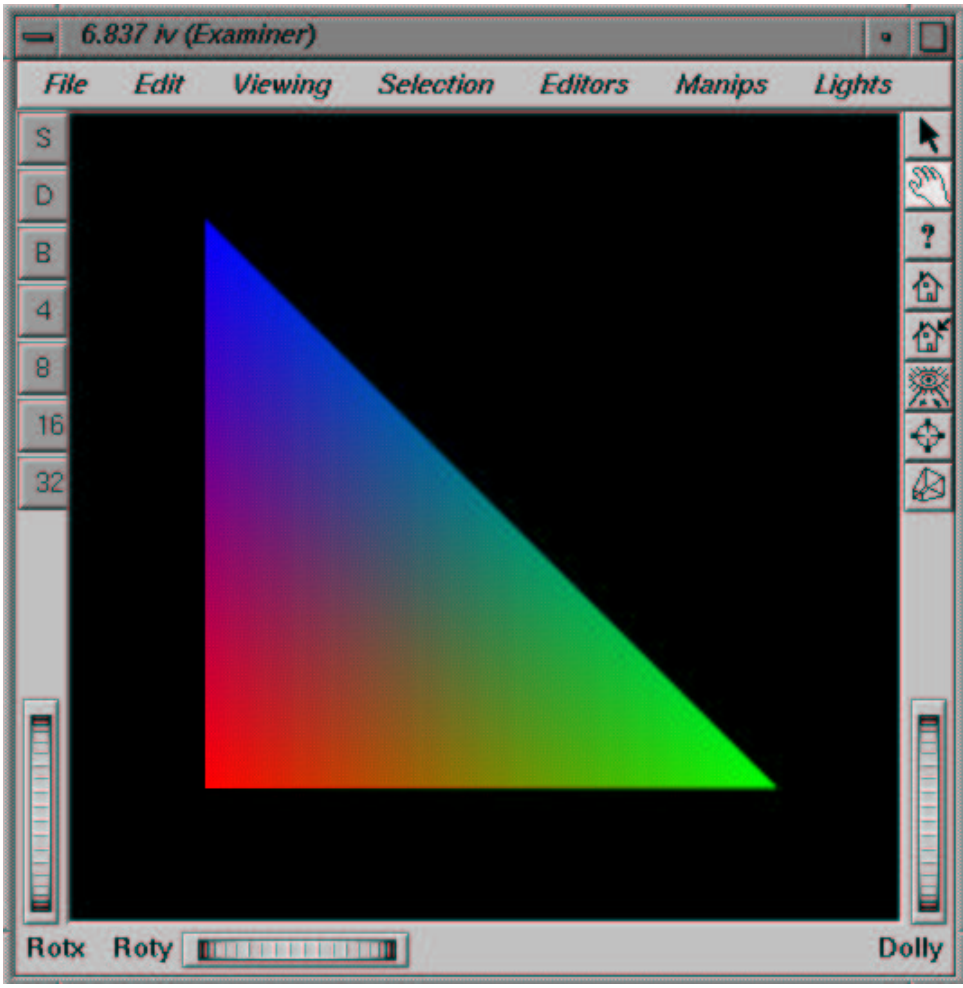
If an object in shadow appears to have a certain luminanc

system compensates for shadow by interpreting object's

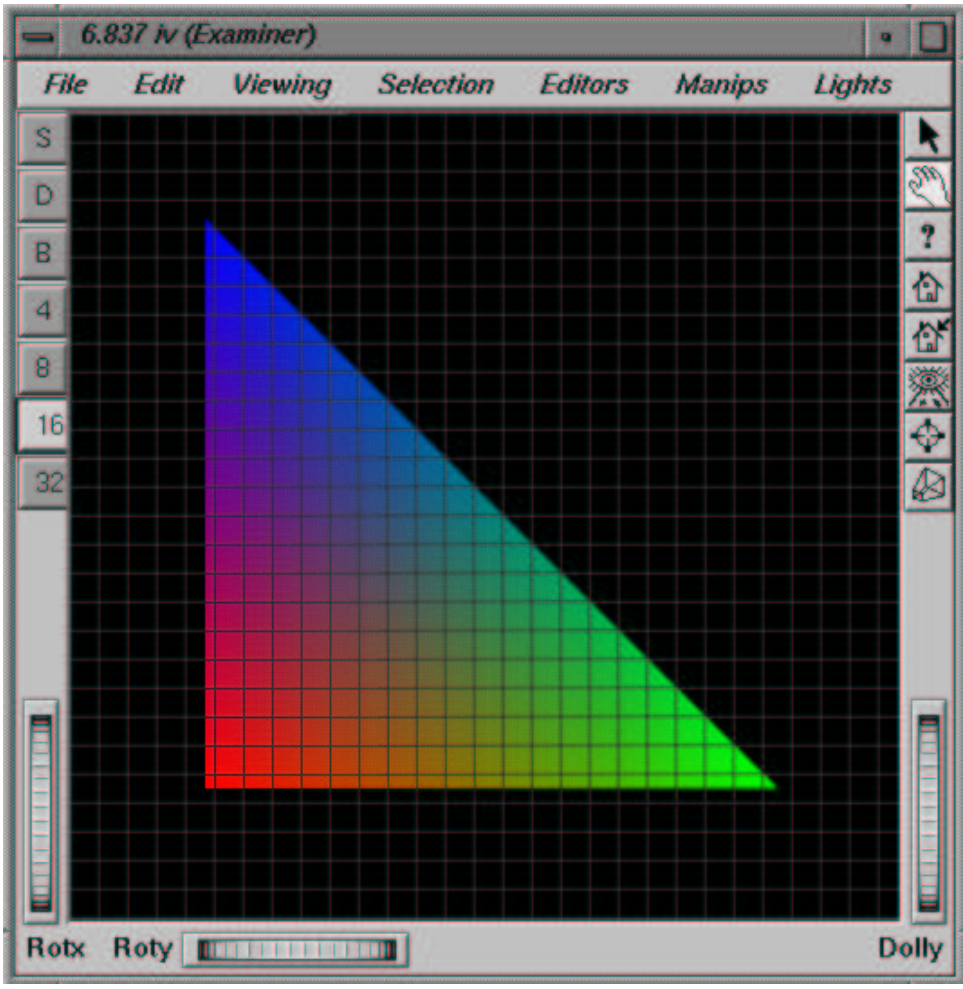
Full explanation on Prof. Edelson's site.

Assignment 4: `ivscan`

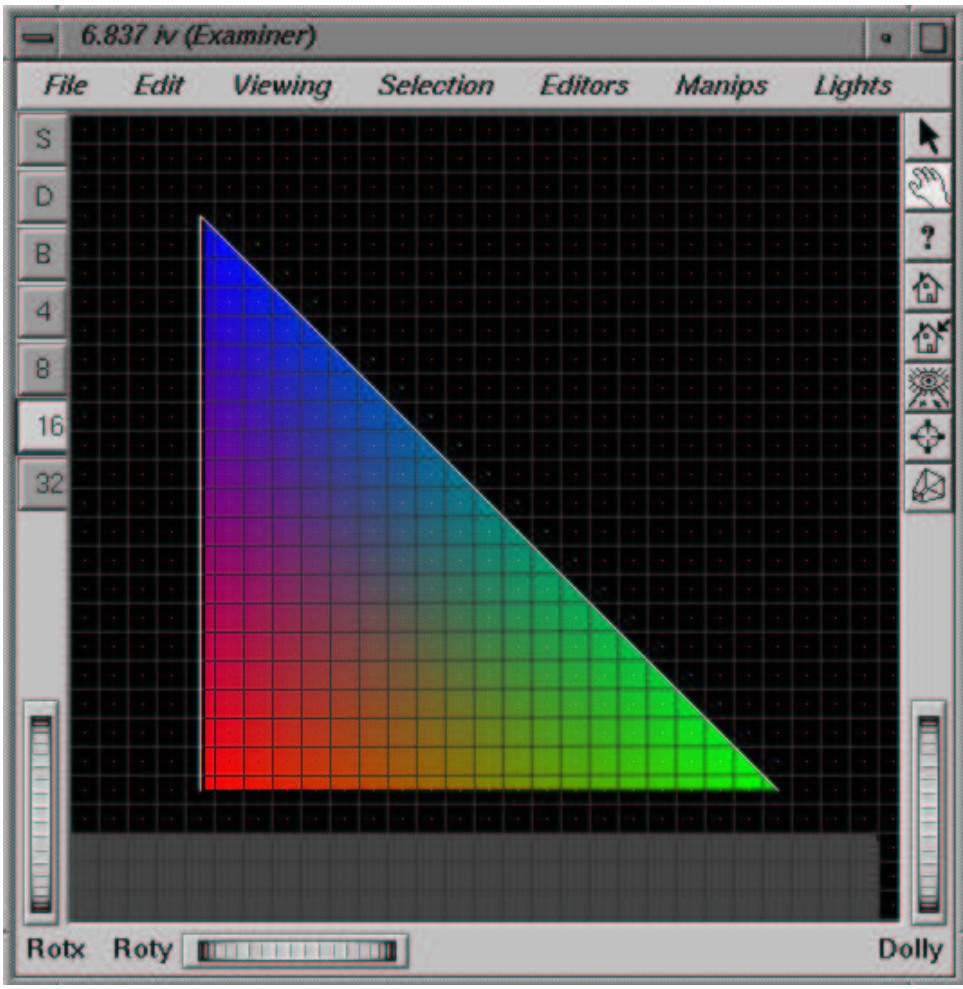
Inventor input:



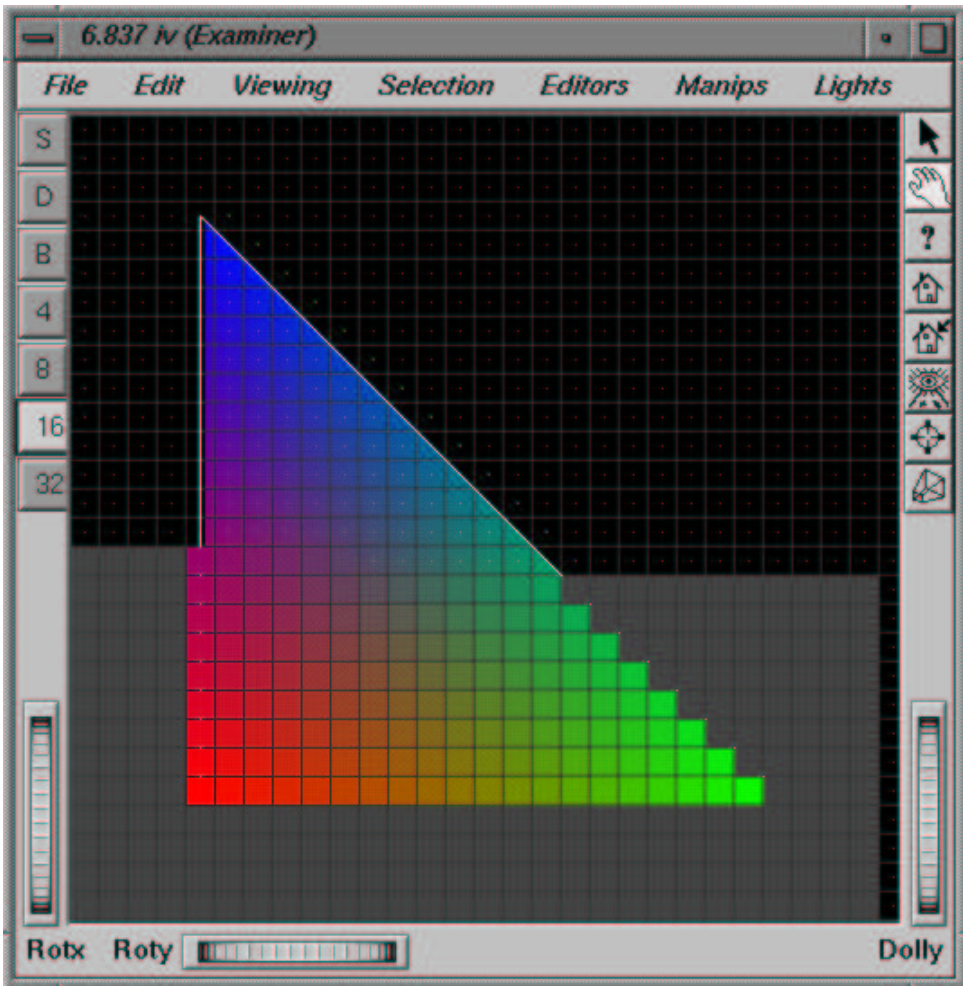
Pixels outlined:



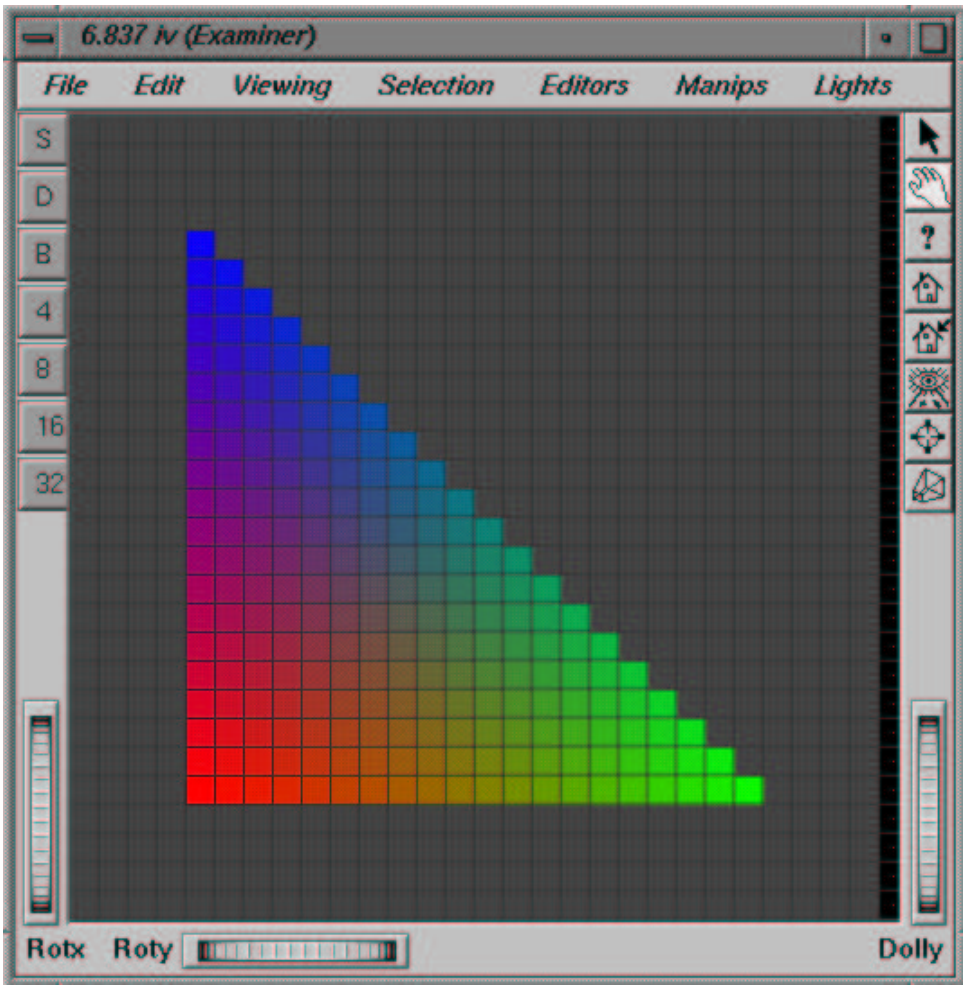
Edges outlined:



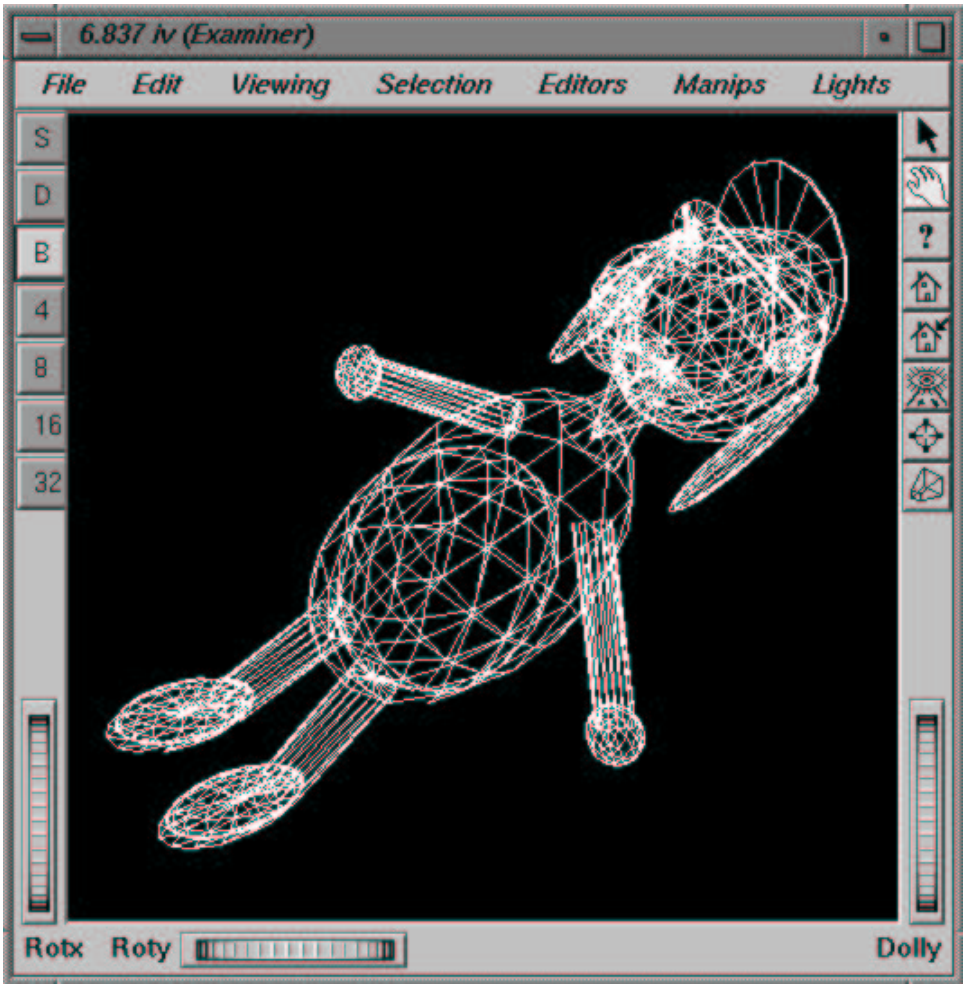
Scan conversion [calls to `setPixel()`]:



Resulting image:



Another Example



Scan conversion [calls to `Raster.setPixel()`]: