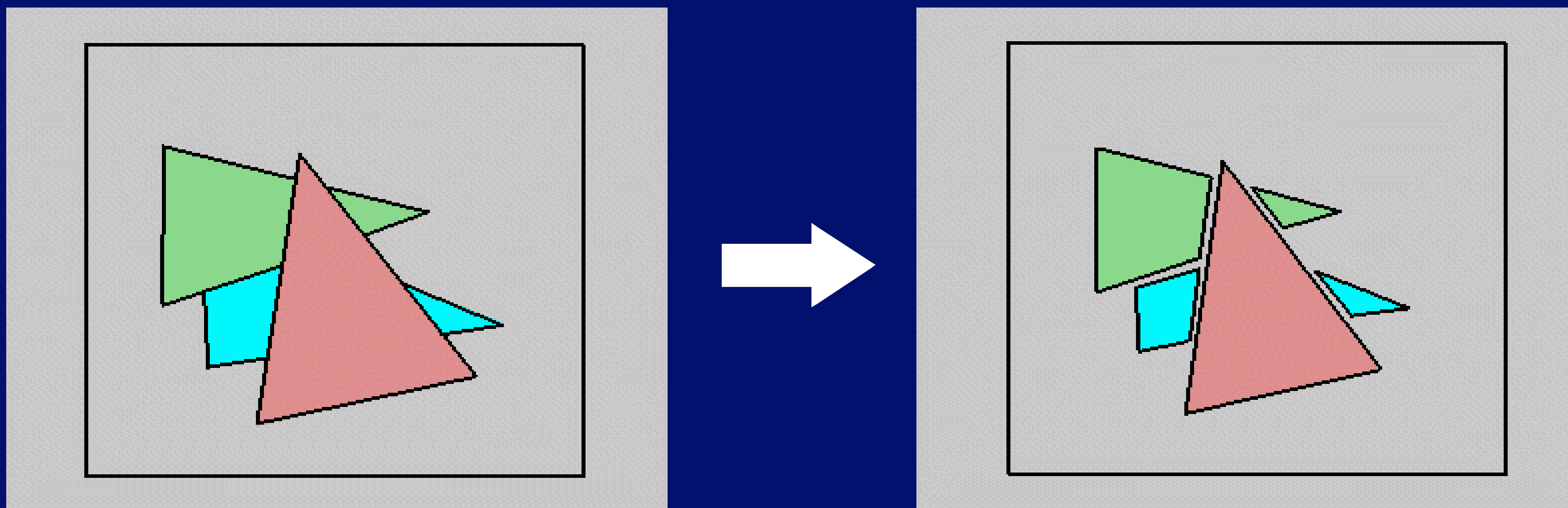
CS 445 / 645
Introduction to Computer Graphics

Lecture 21

Visibility

Painter's Algorithm

Simple approach: render the polygons from back to front, "painting over" previous polygons:

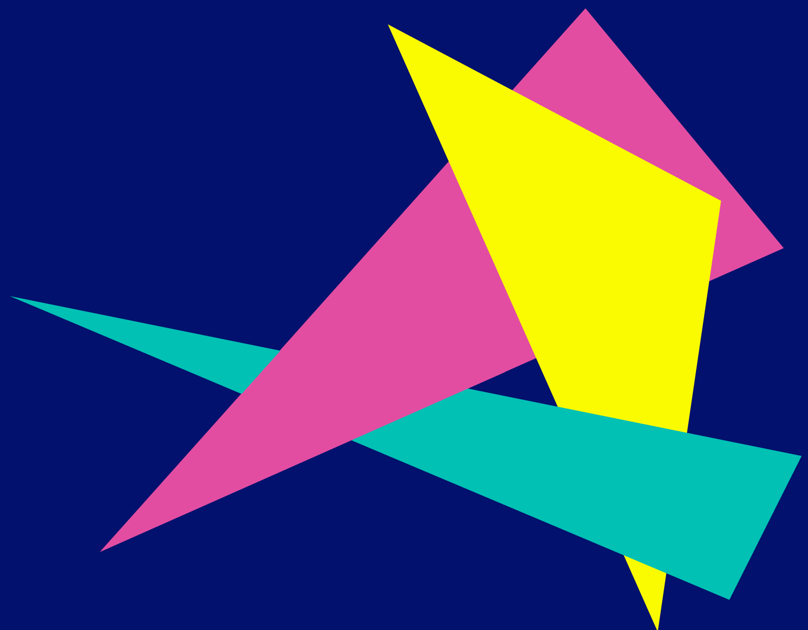


- Draw blue, then green, then orange

Will this work in the general case?

Painter's Algorithm: Problems

- **Intersecting polygons** ***present a problem***
- ***Even non-intersecting polygons can form a cycle with no valid visibility order:***



Binary Space Partition Trees (1979)

BSP tree: organize all of space (hence partition) into a binary tree

- *Preprocess*: overlay a binary tree on objects in the scene
- *Runtime*: correctly traversing this tree enumerates objects from back to front
- Idea: divide space recursively into half-spaces by choosing *splitting planes*
  - Splitting planes can be arbitrarily oriented

Polygons:
BSP Tree Construction

Split along the plane defined by any polygon from scene

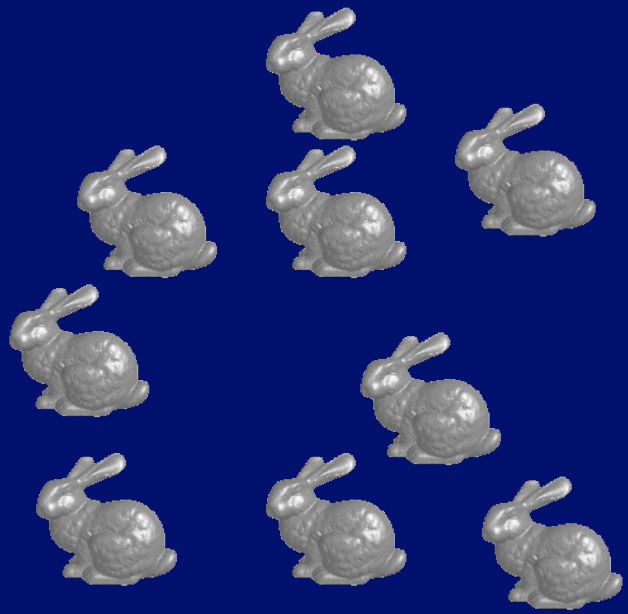
Classify all polygons into positive or negative half-space of the plane

- If a polygon intersects plane, split polygon into two and classify them both

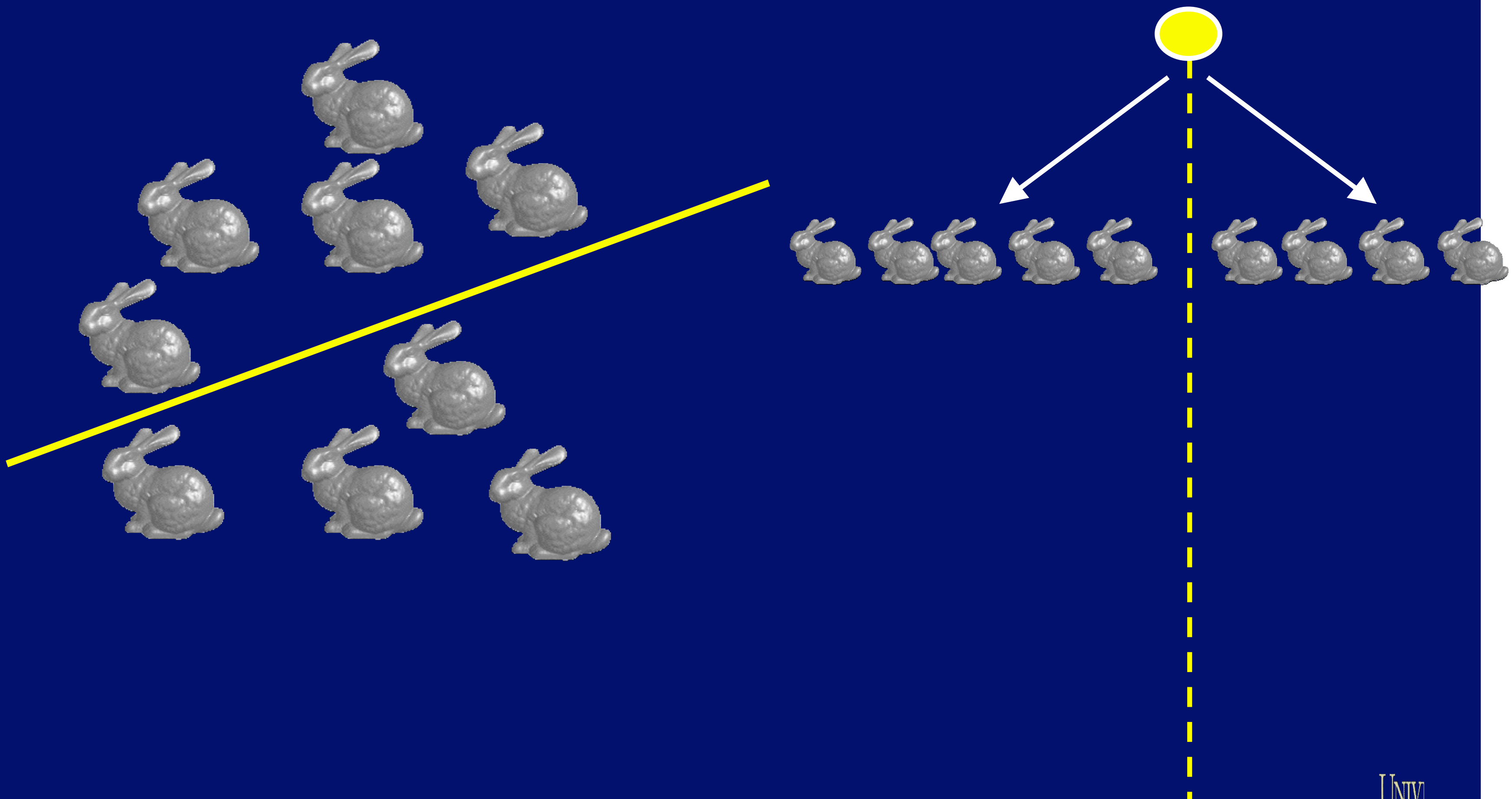
Recurse down the negative half-space

Recurse down the positive half-space

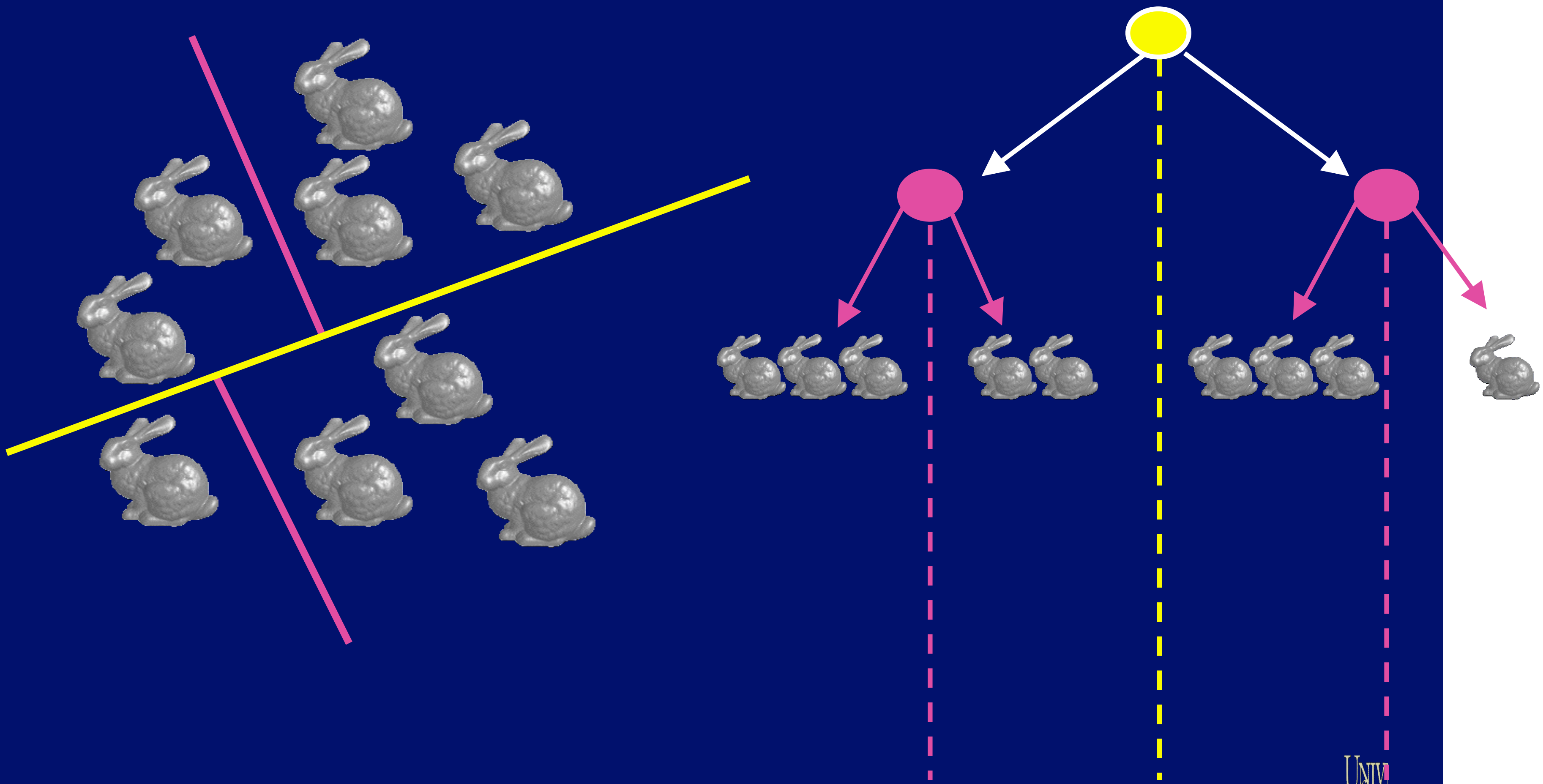
BSP Trees: Objects



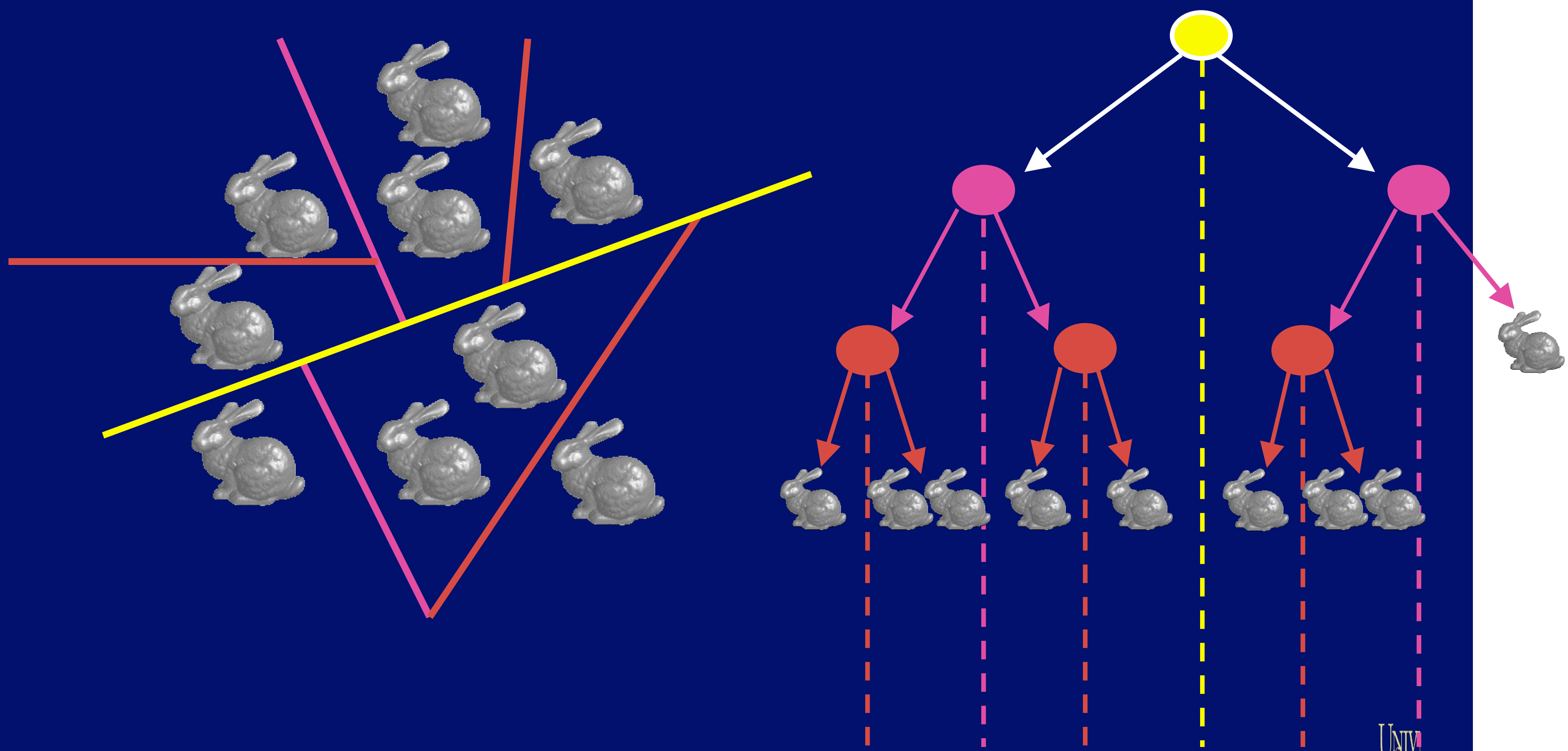
BSP Trees: Objects



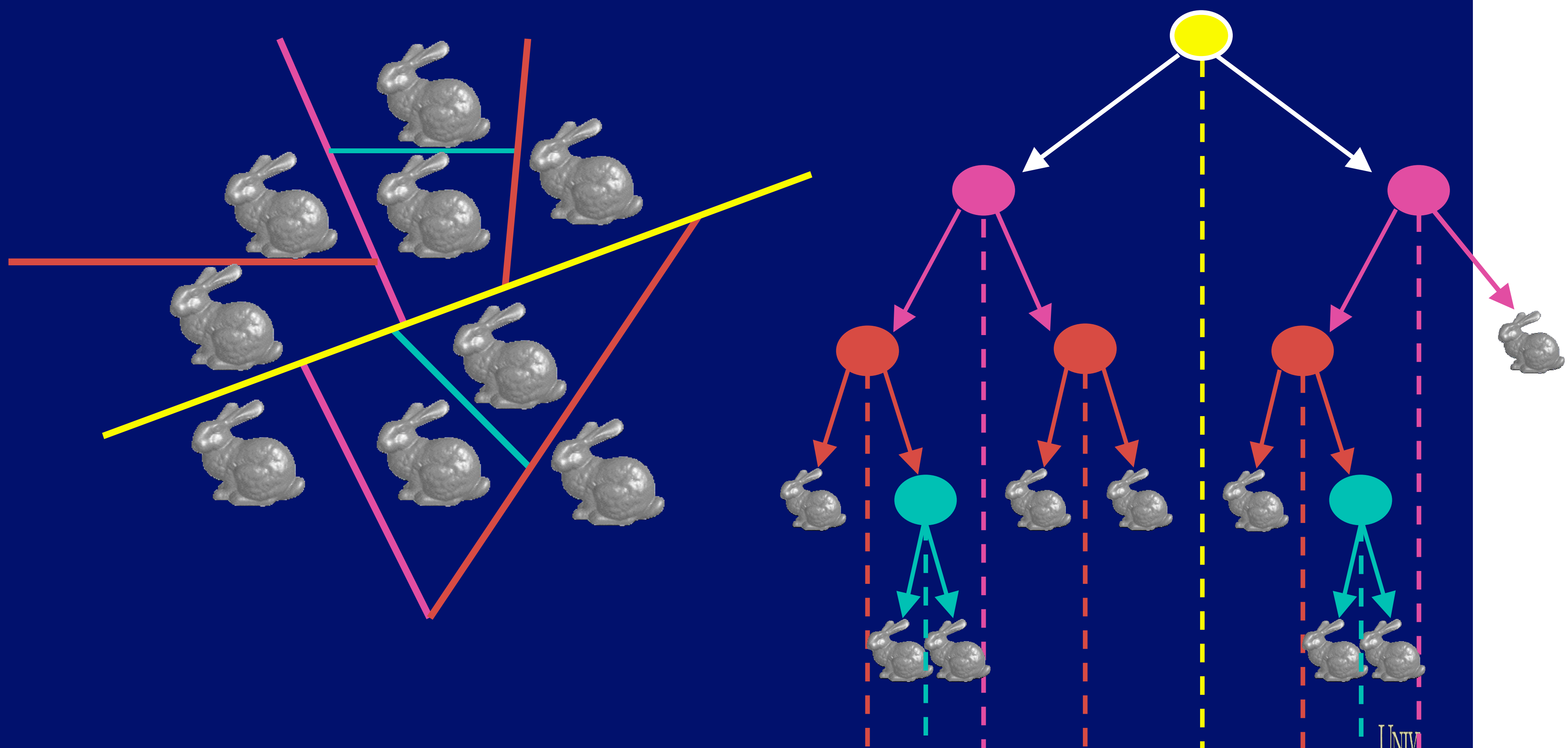
BSP Trees: Objects



BSP Trees: Objects



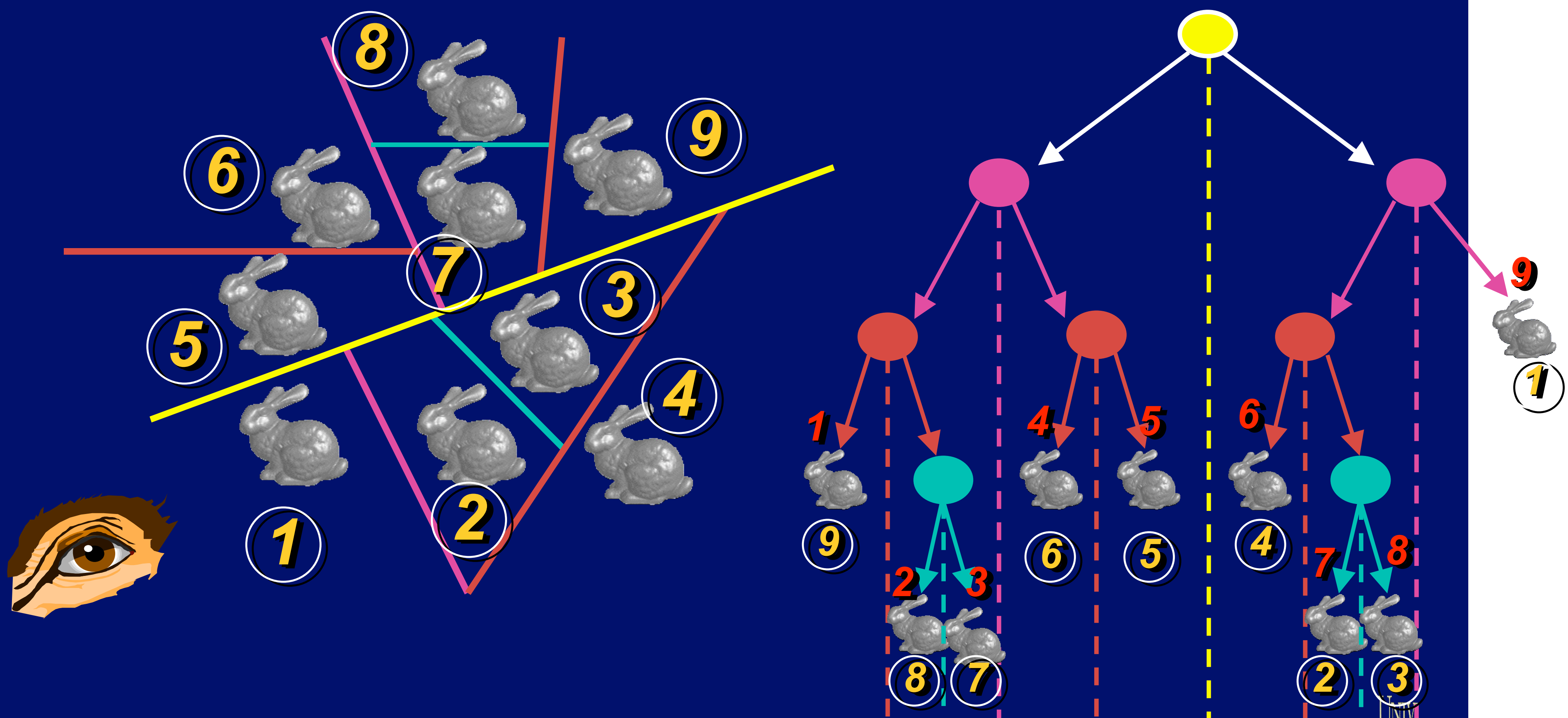
BSP Trees: Objects



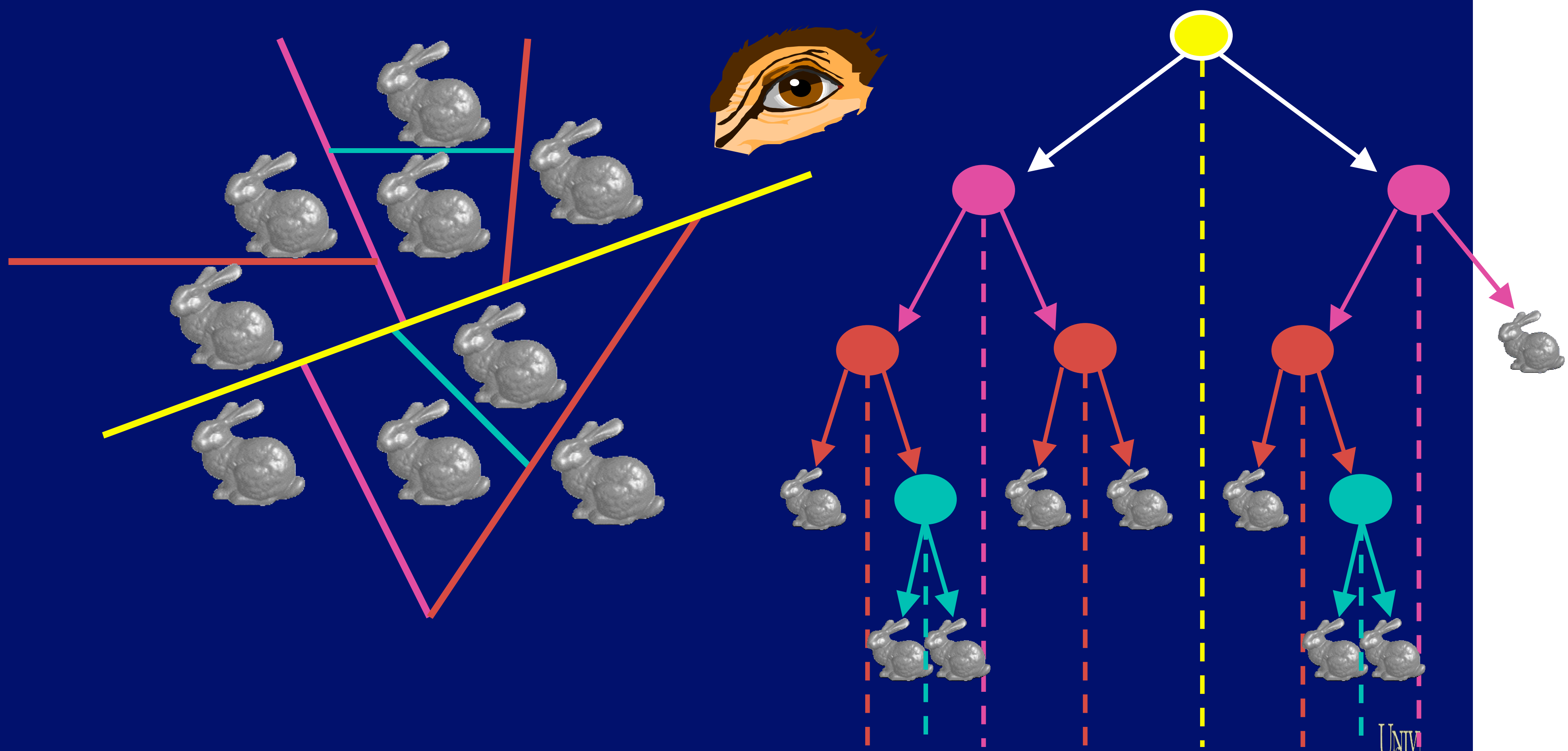
Rendering BSP Trees

```
renderBSP(BSPtree *T)
  BSPtree *near, *far;
  if (eye on left side of T->plane)
      near = T->left; far = T->right;
  else
      near = T->right; far = T->left;
  renderBSP(far);
  if (T is a leaf node)
      renderObject(T)
  renderBSP(near);
```

Rendering BSP Trees

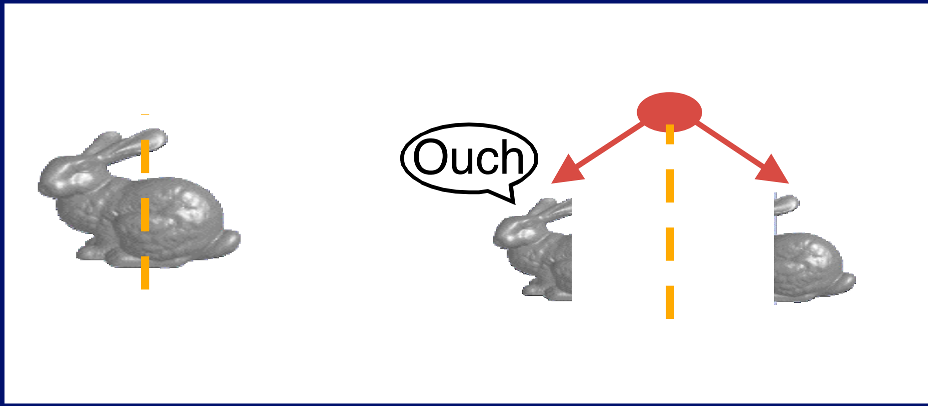


Rendering BSP Trees



Discussion: BSP Tree Cons

- ***No bunnies were harmed in my example***
- ***But what if a splitting plane passes through an object?***
  - Split the object; give half to each node



BSP Demo

Nice demo:

- http://symbolcraft.com/graphics/bsp

Summary: BSP Trees

Pros:

- Simple, elegant scheme
- Only writes to framebuffer (no reads to see if current polygon is in front of previously rendered polygon, i.e., painters algorithm)
  - Thus very popular for video games (but getting less so)

Cons:

- Computationally intense preprocess stage restricts algorithm to static scenes
- Slow time to construct tree
- Splitting increases polygon count

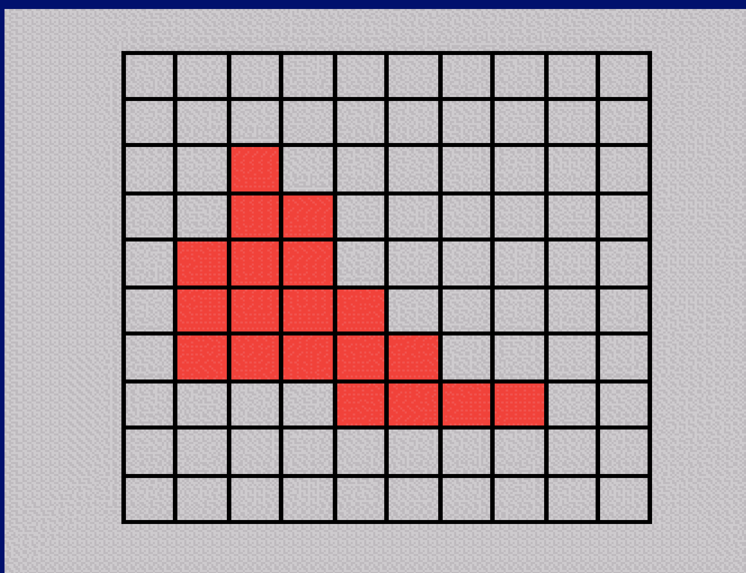
The Z-Buffer Algorithm

- ***BSP trees were proposed when memory was expensive***
  - Example: first 512x512 framebuffer > $50,000!
- ***Ed Catmull (mid-70s) proposed a radical new approach called* z-buffering.**
- ***The big idea: resolve visibility* independently at each pixel**

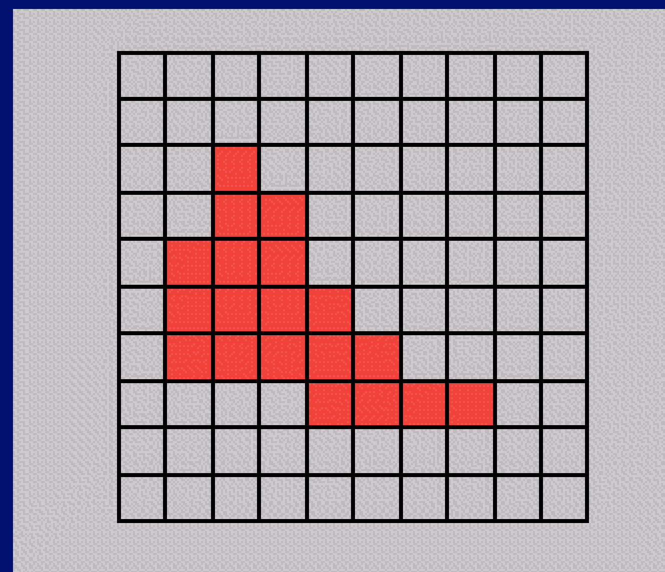
The Z-Buffer Algorithm

- *We know how to rasterize polygons into an image discretized into pixels:*

Color buffer

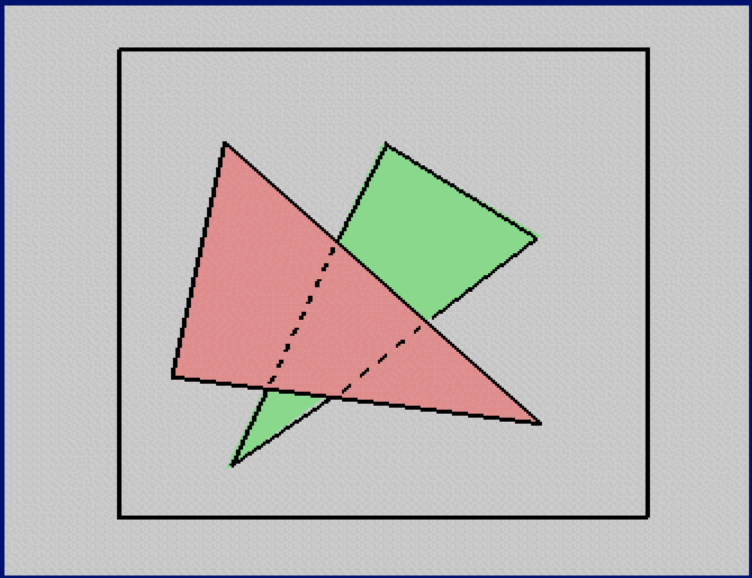


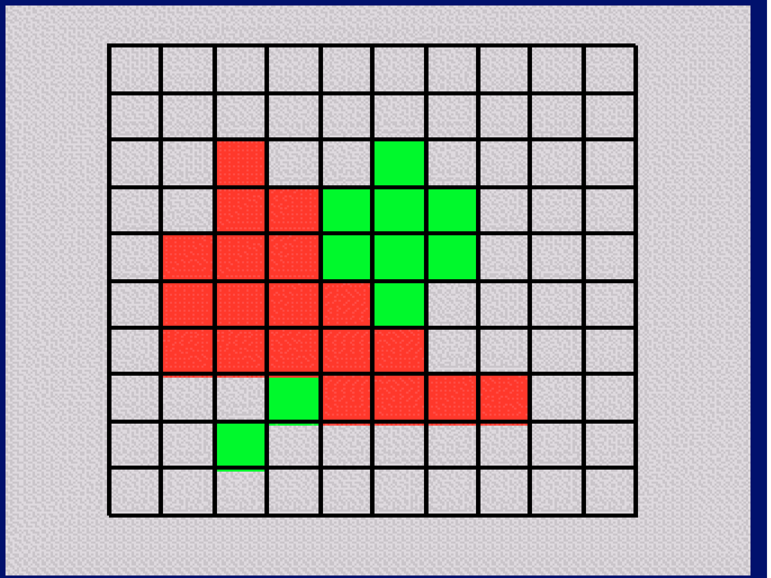
Z buffer



The Z-Buffer Algorithm

What happens if multiple primitives occupy the same pixel on the screen? Which is allowed to paint the pixel?





The Z-Buffer Algorithm

Idea: retain depth (Z in eye coordinates) through projection transform

- Use canonical viewing volumes
- Each vertex has z coordinate (relative to eye point) intact

The Z-Buffer Algorithm

Augment framebuffer with* Z-buffer *or* depth buffer *which stores Z value at each pixel

- At frame beginning, initialize all pixel depths to
- When rasterizing, interpolate depth (Z) across polygon and store in pixel of Z-buffer
- Suppress writing to a pixel if its Z value is more distant than the Z value already stored there

Interpolating Z

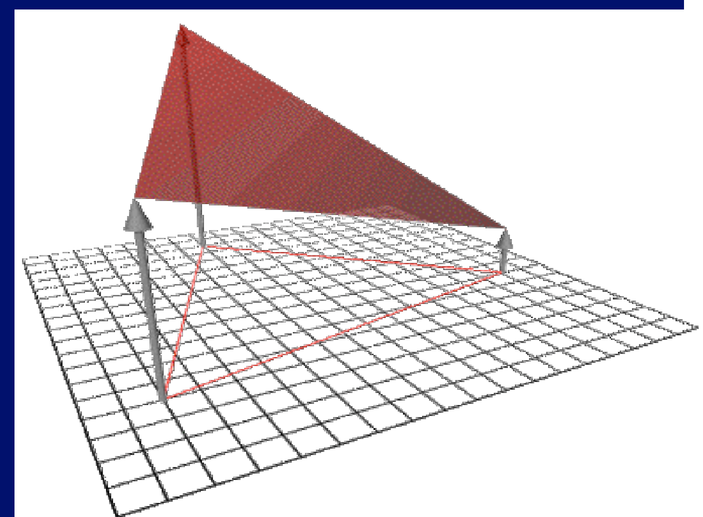
Edge equations: Z is just another planar parameter:

$$z = (-D - Ax - By) / C$$

If walking across scanline by (Δx)

$$z_{new} = z_{old} - (A/C)(\Delta x)$$

- Look familiar?
- Total cost:
  - 1 more parameter to increment in inner loop
  - 3x3 matrix multiply for setup



Edge walking: just interpolate Z along edges and across spans

Z-Buffer Pros

- *Simple!!!*
- *Easy to implement in hardware*
- *Polygons can be processed in arbitrary order*
- *Easily handles polygon interpenetration*
- *Enables* **deferred shading**
  - Rasterize shading parameters (e.g., surface normal) and only shade final visible fragments

Z-Buffer Cons

Lots of memory (e.g. 1280x1024x32 bits)

- With 16 bits cannot discern millimeter differences in objects at 1 km distance

Read-Modify-Write in inner loop requires fast memory

Hard to do analytic antialiasing

- We don't know which polygon to map pixel back to

Shared edges are handled inconsistently

- ***Ordering dependent***

Hard to simulate translucent polygons

- We throw away color of polygons behind closest one

GPU vs. CPU

GPU

- Graphics processing Unit, introduced in 1999 for the PC industry
- Technically defined a single chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is capable of processing a minimum of 10 million polygons per second." We don't know which polygon to map pixel back to
- ***With the advent of the GPU, computationally intensive transform and lighting calculations were offloaded from the CPU onto the GPU—allowing for faster graphics processing speeds.***
- ***It lends itself to some problems involving very costly computation.***