

# Courseware on Hidden Line Surface Removal

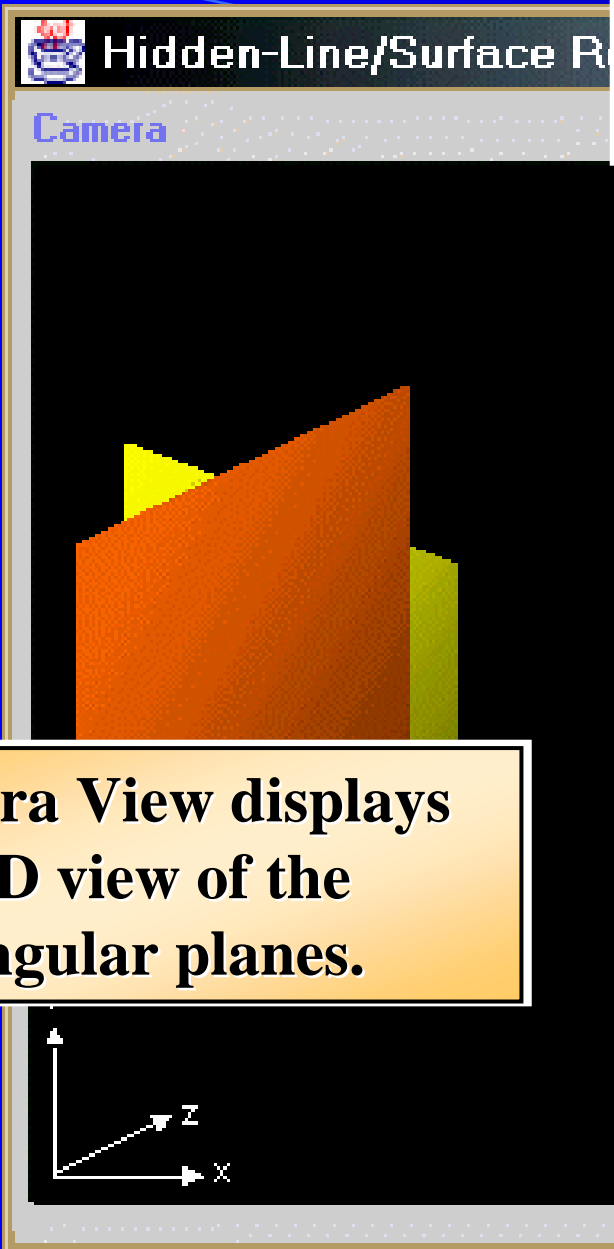
by T.Cheah, C.W. Tan, and P. Malone

# Why Develop This Courseware?

- As a teaching aid for this topic.
- Provide a better visualization to the each different algorithm.
- Gain better understanding through experimenting and interaction.
- Compare and contrast the feature of each algorithm.

# How Is It Done?

- Code our own 3-D engine.
- Implements the three algorithms: Painter's, Z-Buffer, and Depth Coherence (Warnock).
- Rectangular planes can be placed by user freely anywhere in the 3-D world.
- The Camera View renders the planes using the specified algorithm.
- The Top View provides a better view to the orientation of all the planes.



**The chosen Hidden Surface Removal algorithm used in rendering.**

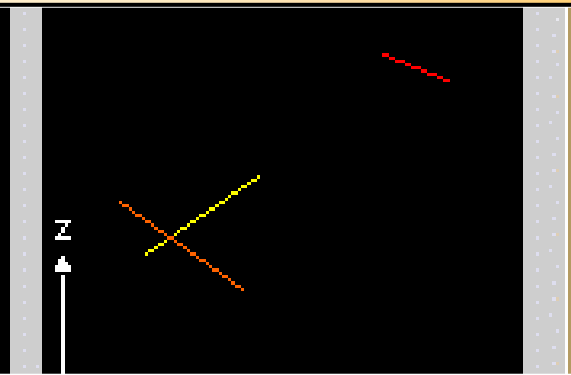
Painter's Algorithm

**Polygons List:**

- Green
- Red
- Yellow
- Orange

**Results of the Algorithm**

**Camera View displays the 3-D view of the rectangular planes.**



**Top View displays the layout and orientation of all the planes.**

**Hidden-Line/Surface Removal Algorithm**

Camera Algorithm:

Painter's Algorithm  
Painter's Algorithm  
Z-Buffer Algorithm  
Depth Coherence Algorithm

Yellow

**Choose different algorithm**

Clear Add Blue

The image shows a software window titled "Hidden-Line/Surface Removal Algorithm". It has a "Camera" tab and an "Algorithm:" dropdown menu. The dropdown menu is open, showing "Painter's Algorithm" selected, with other options being "Painter's Algorithm", "Z-Buffer Algorithm", and "Depth Coherence Algorithm". Below the menu is a text input field containing "Yellow". A yellow button labeled "Choose different algorithm" is positioned below the text field. To the right of the button is a 2D depth map visualization with a grid of colored pixels and several lines. At the bottom of the window are three buttons: "Clear", "Add", and a color selection button currently set to "Blue". On the left side of the window, there is a 3D scene with several colored planes (orange, yellow, red, green) and a 3D coordinate system with axes labeled x, y, and z.

**Hidden-Line/Surface Removal Algorithm**

Camera Algorithm:

Painter's Algorithm  
 Painter's Algorithm  
**Z-Buffer Algorithm**  
 Depth Coherence Algorithm

Observe the Camera View ...

Clear Add Blue

The image shows a software window titled "Hidden-Line/Surface Removal Algorithm". It has two tabs: "Camera" and "Algorithm". The "Algorithm" tab is active, showing a list of algorithms: "Painter's Algorithm", "Painter's Algorithm", "Z-Buffer Algorithm" (highlighted), and "Depth Coherence Algorithm". Below the list is a yellow button labeled "Observe the Camera View ...". To the right of the button is a 2D visualization of a depth buffer, showing a grid of colored pixels and several lines. Below this visualization are three buttons: "Clear", "Add", and "Blue". In the "Camera" tab, a 3D scene is visible with several colored planes (orange, yellow, red, green) and a 3D coordinate system with axes labeled x, y, and z.

**Hidden-Line/Surface Removal Algorithm**

Camera Algorithm:

Z-Buffer Algorithm

Green  
Red  
Yellow  
Orange

See any difference ??

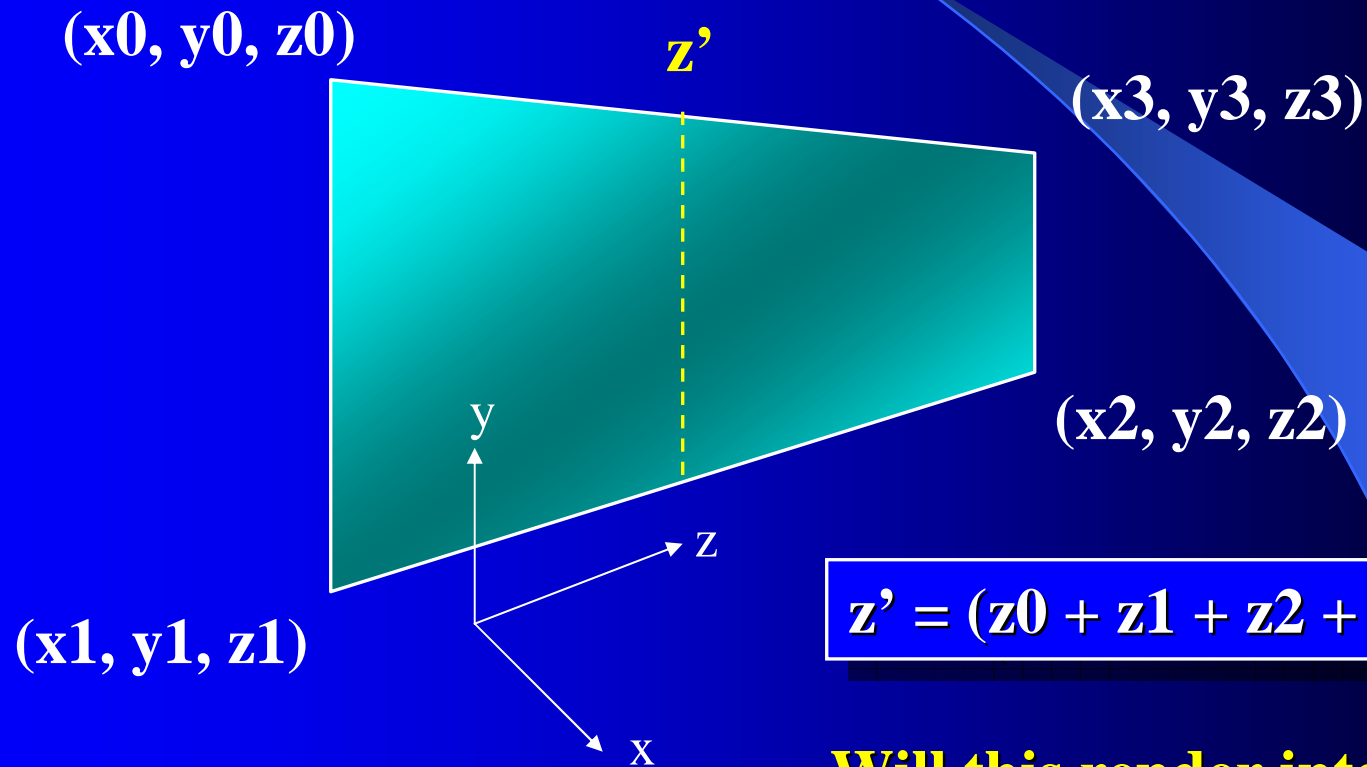
Clear Add Blue

# The Painter's Algorithm

- The simplest Hidden Surface Removal algorithm.
- Sort all the planes from the farthest to the nearest.
- Project them in the sorted order.

**A plane has four vertices, which Z-value of the vertex should be chosen for sorting ??**

The best approach will be using the average Z-value of all vertices.



$$z' = (z_0 + z_1 + z_2 + z_3) / 4$$

**Will this render intersecting planes properly??**

# Z - buffer Algorithm

- Also known as the Depth Buffer Algorithm
- Second Simplest to the Painters Algorithm
- Implemented in hardware
  - slow in software because its implemented for each pixel of each polygon

# Z - buffer Implementation

- Use 2 matrices
  - $I[x,y]$  is the intensity buffer
  - $D[x,y]$  is the depth buffer
- Initialize
  - $I[x,y]$  to the background color of the canvas
  - $D[z,y]$  to infinity (I used negative one)

# Z - buffer Algorithm

- For each Polygon (the order is unimportant)
  - For each pixel in that polygon
    - determine the x and y of that polygon when it is projected (determined through the 3D renderer)
      - These values become the indices into the matrices
    - If the  $z < D[x,y]$  then
      - replace  $D[x,y]$  with  $z$
      - replace  $I[x,y]$  with the polygon color

# Example

- Depth

- |    |    |    |    |
|----|----|----|----|
| -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 |

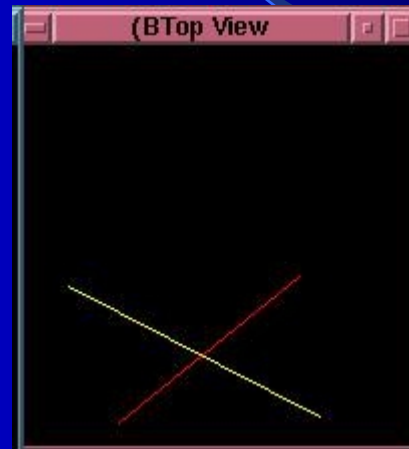
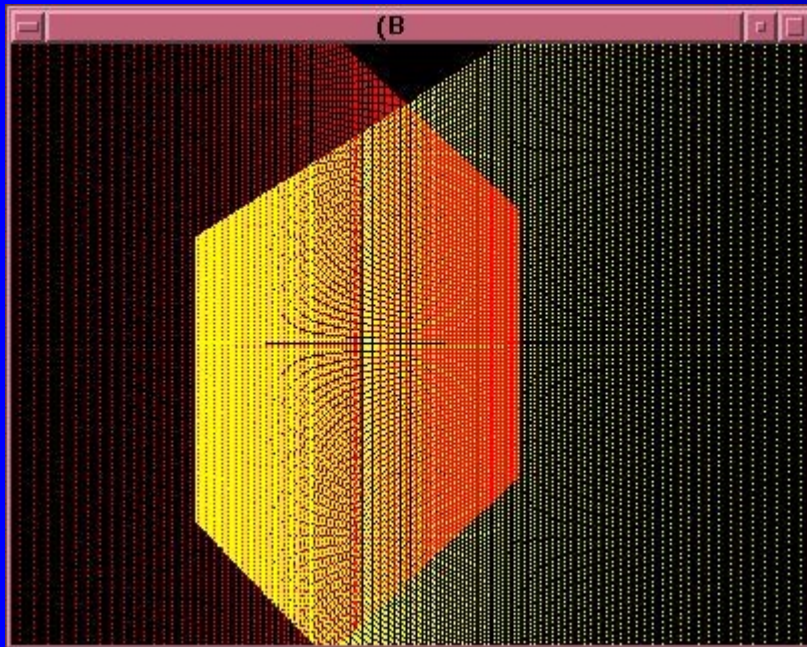
- |    |    |    |    |
|----|----|----|----|
| -1 | -1 | -1 | -1 |
| -1 | 2  | 2  | -1 |
| -1 | -1 | -1 | -1 |

## Intensity

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 |

# So far



Nov 30, 2001

Courseware on Hidden Line/  
Surface Removal

# Scan Line Algorithm

- Based On the z - buffer algorithm
- Uses scan lines instead of going pixel by pixel

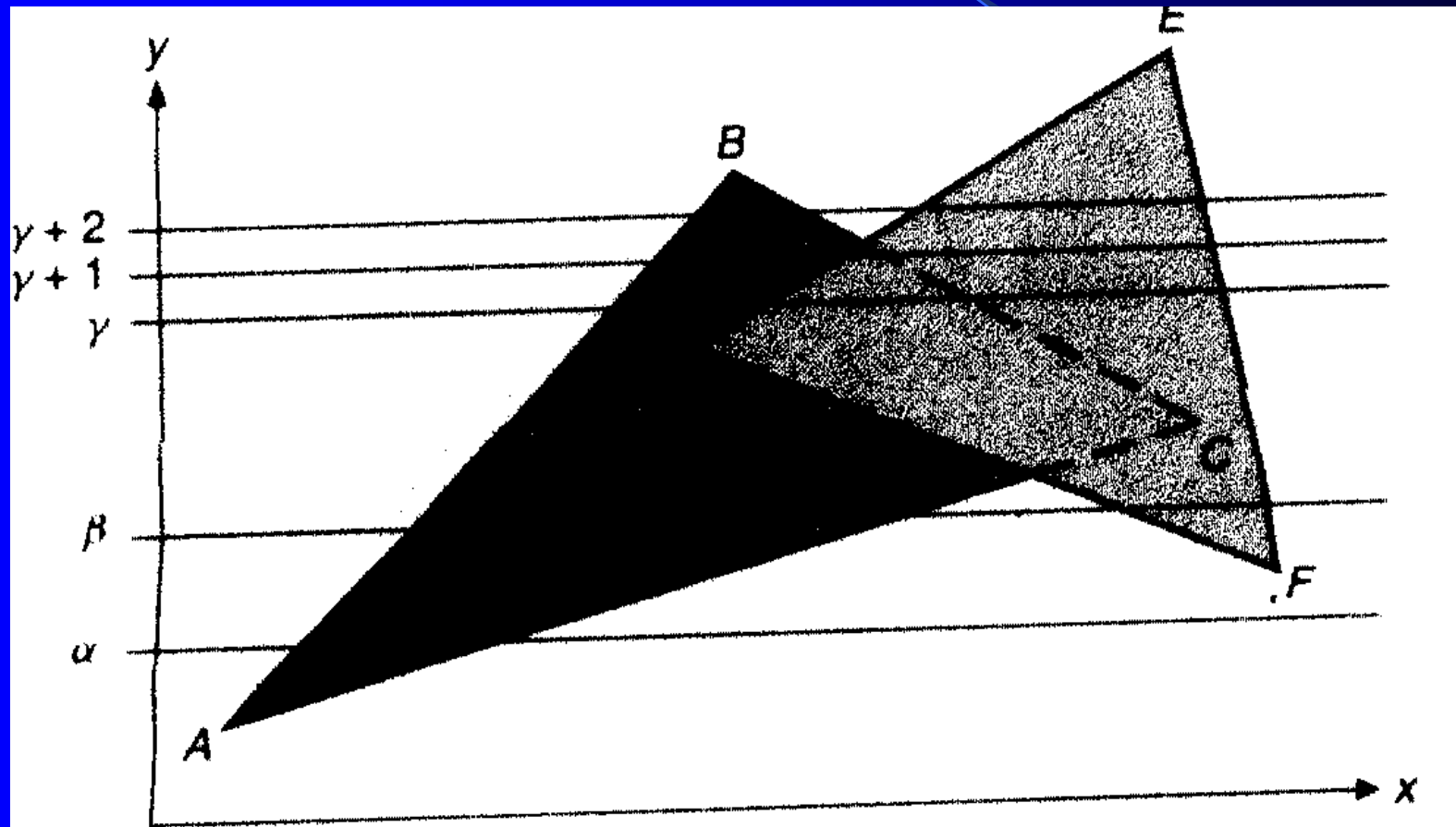
# Scan Line Algorithm

- Use 2 arrays
  - $D[x]$  distance array
  - $I[x]$  intensity array
- Use a scan line along each  $y$  instead of 2 dimensional arrays

# Scan Line Algorithm

- For each Polygon find all the pixels that are on the current scan line
- For each of these pixels
  - if  $z < D[x]$  then
    - $D[x] = z$
    - $I[x] =$  the polygon color
- After all scan lines and all polygons the I array contains the color for each pixel of the scene

# Example



Nov 30, 2001

Courseware on Hidden Line/  
Surface Removal

# Depth Coherence Algorithm

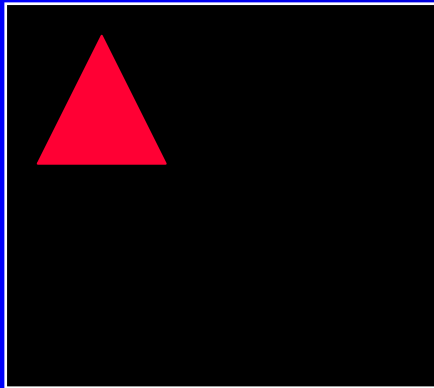
- Also called Warnock's Area-Subdivision Algorithm
- Subdivide areas into pieces until they can be trivially solved.
- A divide and conquer Algorithm
  - Recursive implementation

# Depth Co. Algorithm (cont.)

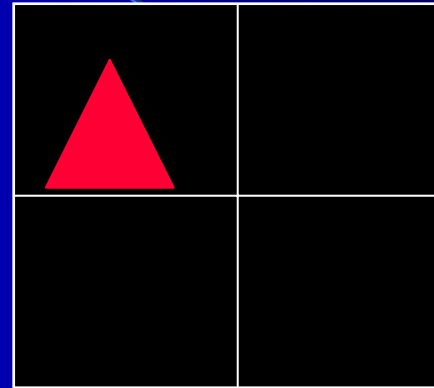
- 3 base cases
  - Area is in **pixel size**: choose the closest polygon's color
  - Area **disjoint** with all polygons: paint area with background color
  - Area **surrounded** by the closest polygon : paint area with the polygon's color

# How?

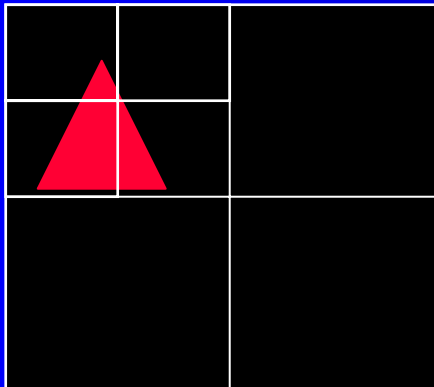
1



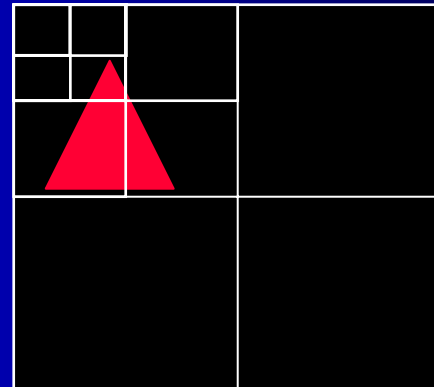
2



3

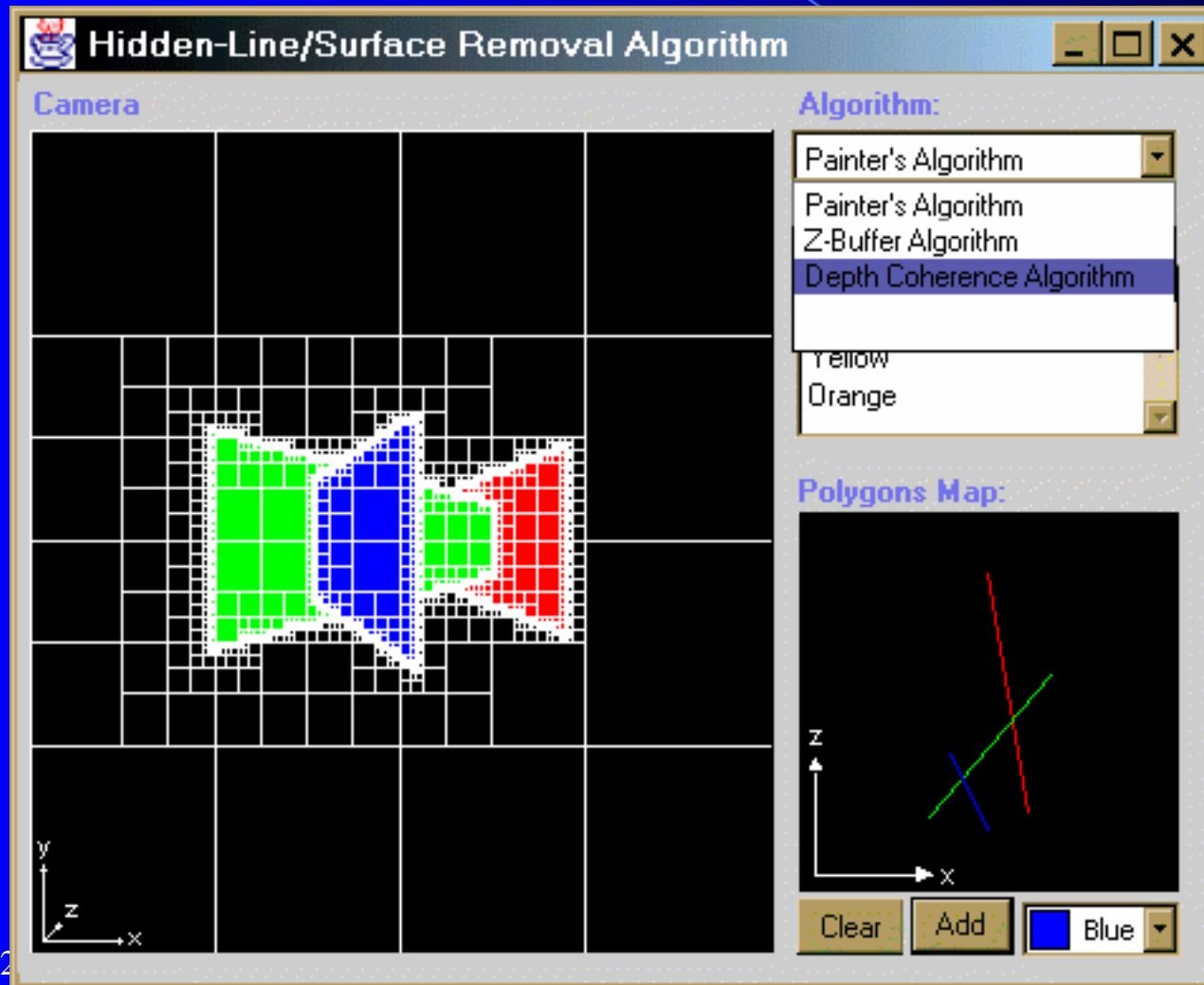


4



**and so on...**

# Preview



Nov 30, 2002

Surface Removal

# Features

- **Image precision, yet object precision for depth comparison**
- **Able to draw scene with intersecting polygons**
- **Fast for scene with big polygons**

# Downside

- For complex scene, many redundant & overlapping areas (rectangles)
  - possibly many pixel level cases
  - worse than z-buffer algorithm
- Recursive implementation
  - Large memory consumption

# Questions ??



Nov 30, 2001

Courseware on Hidden Line/  
Surface Removal



Thanks

Nov 30, 2001

Courseware on Hidden Line/  
Surface Removal