Visible Surface Determination

Drawing Algorithm

1. Store polygons in a polygon mesh
2. Perform viewplane transformations
3. Perform perspective projection
4. Remove backfaces
5. Decompose polygons into triangles, if necessary
6. Sort the triangles
7. Use painter's algorithm to draw the polygons from back to front

Limitations

1. Single, convex objects
2. Will not work by itself for multiple objects or concave objects
3. A good preprocessor

Decomposition into Triangles

1. Triangles are the simplest polygons

2. Triangles are always convex

3. Tests
   - Minimax test
   - Inside-outside test
   - Test for overlapping triangles

Normal Vectors

Two vectors (a, b, c) and (x, y, z) are said to be normal to each other if

$$\begin{bmatrix} a & b & c \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 0$$

Finding the Side of a Face (1)

- All points (x, y, z) such that

$$Ax + By + Cz = 0$$

  lie on a plane that contains the origin

- Vector (A, B, C) is normal to that plane

- The positive side of the plane looks in the same direction as the normal vector

Finding the Side of a Face (2)

- If (a, b, c) is another vector, then all points

$$(x-a, y-b, z-c)$$

  for which

$$A(x-a)+B(y-b)+C(z-c)=0$$

  lie in another plane normal to (A, B, C)

- If $(x-a, y-b, z-c)$ lies in this plane (normal to (A, B, C)), then (x, y, z) lies in a plane parallel to this one translated by vector (a, b, c)

Finding the Side of a Face (3)

- Thus, $A(x-a)+B(y-b)+C(z-c)=0$ represents a plane through point (a,b,c) and normal to vector (A,B,C)

- We can write this equation as

$$Ax+By+Cz-D=0$$

where

$$D=Aa+Bb+Cc$$

Finding the Side of a Face (4)

- Translate vector

$$(A, B, C)$$

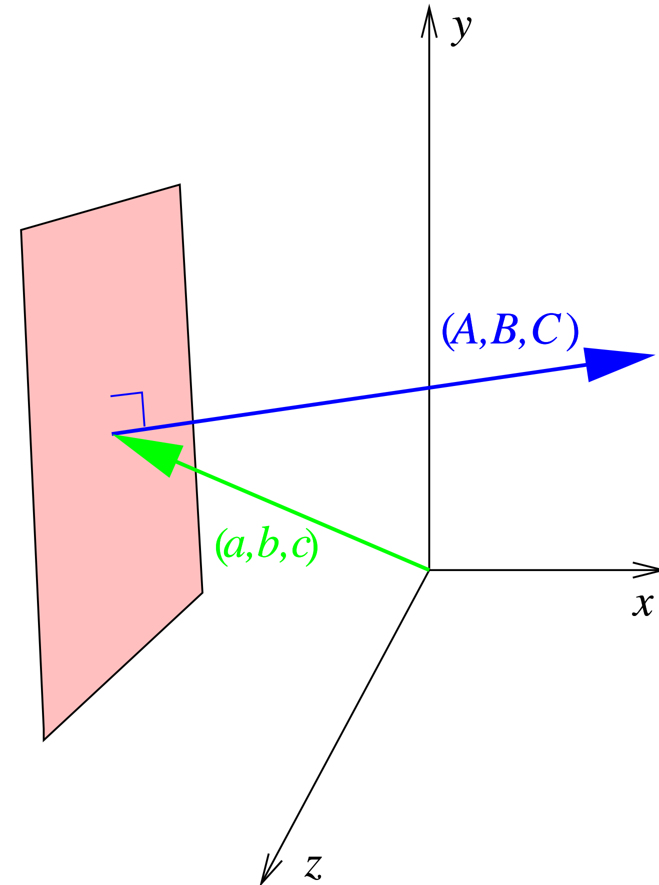
to point

$$(a, b, c)$$

which is on the plane

- The tip is now at

$$(A + a, B + b, C + c)$$



Finding the Side of a Face (5)

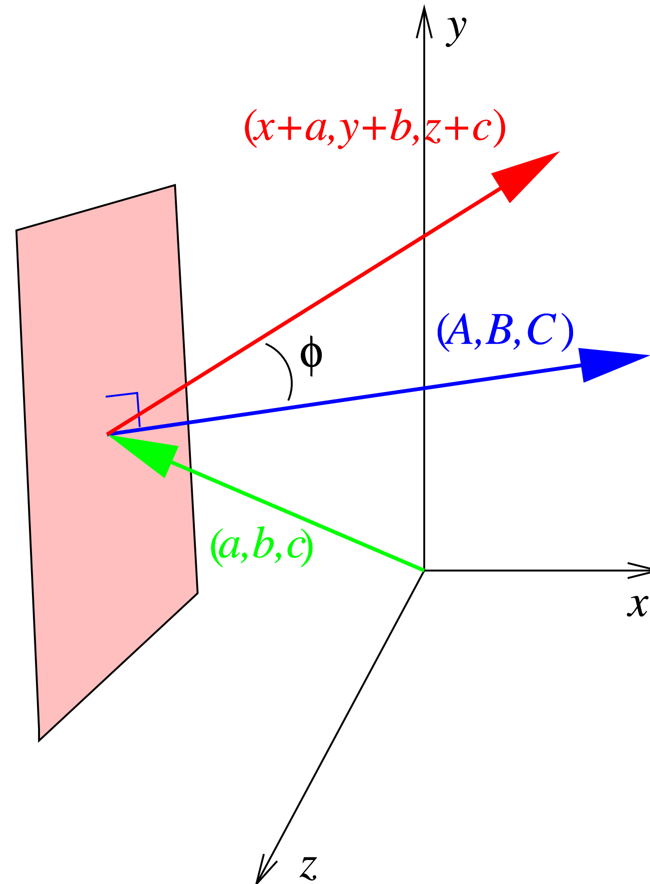
- If a point (x, y, z) lies on the same side of the plane as $(A+a, B+b, C+c)$ (that is, the positive side of the plane), then it can be expressed as a vector

$$(x+a, y+b, z+c)$$

  where the angle between vectors (x, y, z) and (A, B, C) must be less than $\frac{\pi}{2}$

Finding the Side of a Face (6)

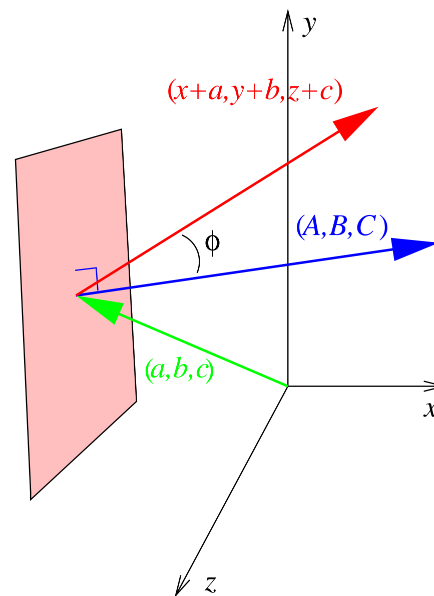
Thus $cos\ \phi > 0$



Finding the Side of a Face (7)

Or,

$$\begin{bmatrix} a & b & c \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \left| \begin{bmatrix} A & B & C \end{bmatrix} \right| \cdot \left| \begin{bmatrix} x & y & z \end{bmatrix} \right| \cdot cos\,\phi > 0$$



Finding the Side of a Face (8)

Insert $(x+a, y+b, z+c)$ into the plane representation:

$$A(x+a)+B(y+b)+C(z+c)-D$$

$$\begin{bmatrix} A & B & C \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + D - D > 0$$

Finding the Side of a Face (9)

$$\begin{bmatrix} A & B & C \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + D - D > 0$$

- This says the plane represntation yields a positive result for every point on the side of the plane that “looks” in the same direction that (A, B, C) points
- It is negative on the other side of the plane

Example 1

- Let $(A,B,C) = (5,-3,1)$ and $(a,b,c) = (0,2,4)$
- Consider the plane through (a,b,c) normal to (A,B,C)
- Its equation is $Ax + By + Cz - D = 0$, where

$$D = (5)(0) + (-3)(2) + (1)(4) = -2$$

so

$$5x - 3y + z + 2 = 0$$

Example 1 (cont.)

- Is $(3, 5, -6)$ on the positive side of the plane?

$$(5)(3) + (-3)(5) + (1)(-6) + 2 = -4$$

- No.

Example 2

- Is $(0, 0, 0)$ on the positive side of the plane?

$$(5)(0)+(-3)(0)+(1)(0)+2 \;=\; 2$$

- Yes.

Example 3

- Is $(A,B,C)+(a,b,c)$ on the positive side of the plane?

$$(A,B,C)+(a,b,c) \ = \ (5,-1,4)$$

$$(5)(5)+(-3)(-1)+(1)(4)+2 \ = \ 34$$

- Yes.

Finding the Plane Equations

Given three non-colinear points, find the coefficients of the plane equation

$$\begin{aligned} P_1 &= (x_1, y_1, z_1) \\ P_2 &= (x_2, y_2, z_2) \\ P_3 &= (x_3, y_3, z_3) \end{aligned}$$

Solve the Linear System

- Solve the linear system

$$Ax_i + By_i + Cz_i - D = 0$$

where $i - 1, 2, 3$

for unknowns A, B, C, and D

- Such a system is always solvable

Finding the Coefficients

Compute the determinants

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \qquad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}$$

$$C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_1 & y_2 & 1 \\ x_1 & y_3 & 1 \end{vmatrix} \qquad D = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_1 & y_2 & z_2 \\ x_1 & y_3 & z_3 \end{vmatrix}$$

Orienting the Plane

- Specify the points

$P_1P_2P_3 \quad P_2P_3P_1 \quad P_3P_1P_2$

- These points are given in the same cyclic order (clockwise)
- Specify counter-clockwise order for a right-handed coordinate system

Recall the Drawing Algorithm

1. Store polygons in a polygon mesh
2. Perform viewplane transformations
3. Perform perspective projection
4. Remove backfaces
5. Decompose polygons into triangles, if necessary
6. Sort the triangles
7. Use painter’s algorithm to draw the polygons from back to front

Depth Sorting Methods

- Simply removing backfaces is not enough
  - Concave objects are a problem
  - Partially hidden objects are a problem
- Depth sorting addresses these issues

Depth Sorting

- Also called *geometric sorting*

- Occlusion-compatible order
  - Overlap of minimum containing rectangle
  - Check for edge intersection
  - If all z values in P_1 are less than P_2, then P_1 is less than P_2
  - Can use “tricks” based on the kinds of objects to draw

Depth Calculation

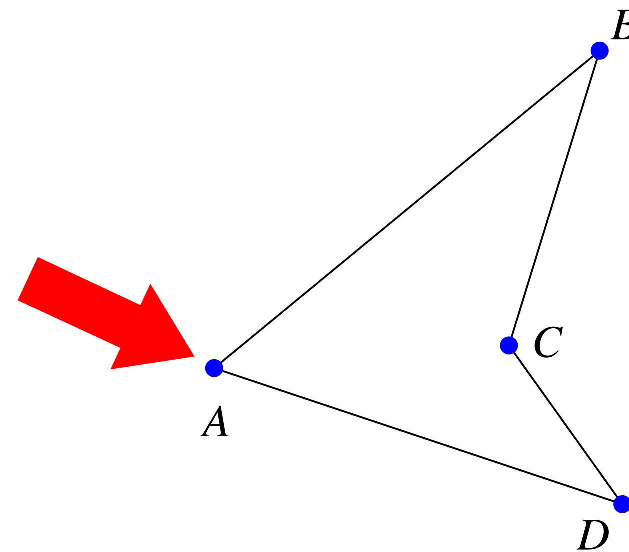
- Depth is easiest to compute with triangular polygons
  - Triangles are the simplest polygons
  - Triangles are always convex

Decomposition into Triangles

- Decomposition is easier for convex polygons, harder for concave polygons
- Choose the leftmost vertex
  - The leftmost vertex *will* be convex
- We are ignoring self-intersecting polygons
  - These generally are not useful for modeling surfaces of solid objects

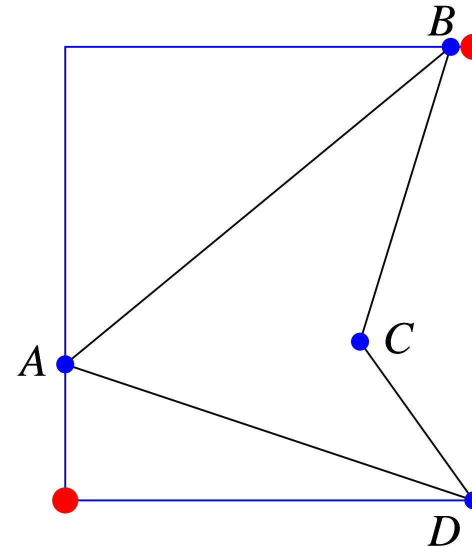
Decomposition

- Choose the leftmost vertex
  - The leftmost vertex has the smallest x value
- Use vertices adjacent to the lefmost vertex to form the first triangle



Decomposition

After forming the first triangle, examine all other vertices to see if any lie within the smallest containing rectangle



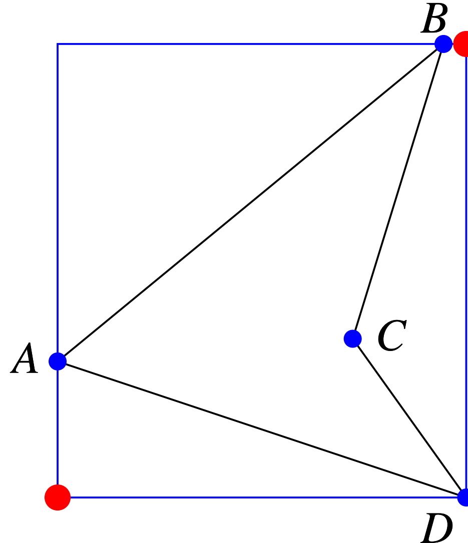
Perform a *minimax* test

- Lower left is at $\mathtt{min}(x_1, x_2, x_3)$, $\mathtt{min}(y_1, y_2, y_3)$
- Upper right is at $\mathtt{max}(x_1, x_2, x_3)$, $\mathtt{max}(y_1, y_2, y_3)$

Minimax Test

The minimax test determines points of interest

$$\left.\begin{array}{lcl} P_x & < & \mathtt{min}(x_i) \\ P_y & < & \mathtt{min}(y_i) \\ P_x & > & \mathtt{max}(x_i) \\ P_y & > & \mathtt{max}(y_i) \end{array}\right\}$$ if any are true, P is outside, so forget it for now



Rearrange the Point-Slope Form of a Line

$$f(x,y) \;=\; (x-x_1)(y_2-y_1)-(x_2-x_1)(y-y_1)$$

Function describes a line passing through two points

$f(x,y)=0 \;\Rightarrow\; (x,y)$ is on the line
$f(x,y)<0 \;\Rightarrow\; (x,y)$ is on one side of the line
$f(x,y)>0 \;\Rightarrow\; (x,y)$ is on the other side of the line

Inside or Outside?

$$f(x,y) = (x - x_1)(y_2 - y_1) - (x_2 - x_1)(y - y_1)$$

Two points lie on the same side of the line if the signs are the same

```
class Point {
    public double x, y, z;
}
boolean sameSide(Point p1, Point p2, Point lpt1, Point lpt2) {
    return (((p1.x - lpt1.x)*(lpt2.y - lpt1.y)
            - (lpt2.x - lpt1.x)*(p1.y - lpt1.y))
            * ((p2.x - lpt1.x)*(lpt2.y - lpt1.y)
            - (lpt2.x - lpt1.x)*(p2.y - lpt1.y)) > 0)? true: false;
}
```

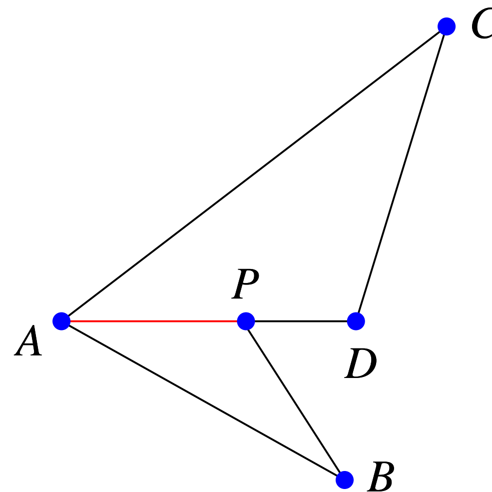
Inside or Outside?

```
boolean inside(Point p, Point a, Point b, Point c) {
    return sameSide(p, a, b, c) && sameSide(p, b, a, c)
            && sameSide(p, c, a, b);
}
```

Inside Test

1. Consider leftmost inside point

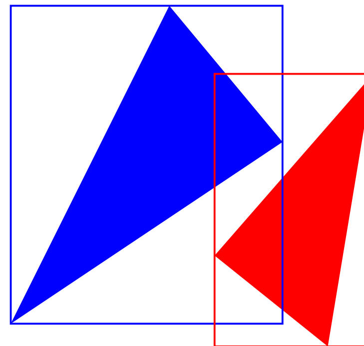
2. Connect original leftmost point to the leftmost inside point
   - If a triangle is formed, put the triangle in the triangle list
   - If one polygon is not a triangle, keep splitting



Tests for Overlapping Triangles

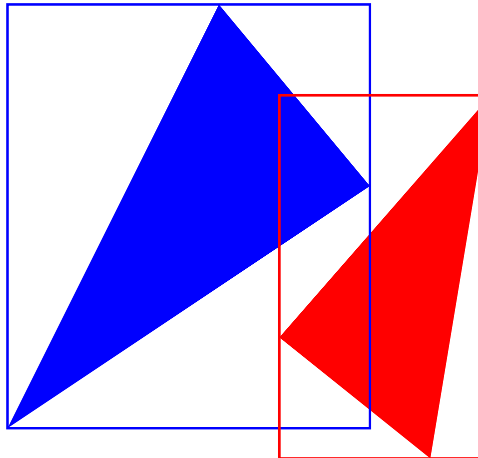
Begin with least expensive tests and move to more expensive tests if necessary

1. Minimax test on smallest containing rectangles
2. Check each edge of T_1 for intersection with each edge of T_2
3. Check for complete containment



Minimax Test on Containing Rectangles

If rectangles don't overlap, neither do the enclosed triangles



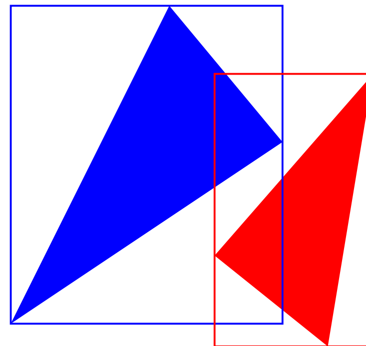
Intersection Tests

Check each edge of T_1 for intersection with each edge of T_2 (use projected points)

1. Minimax tests for lines that make up the edges (cheap)

2. Check for parallel slopes (more expensive)

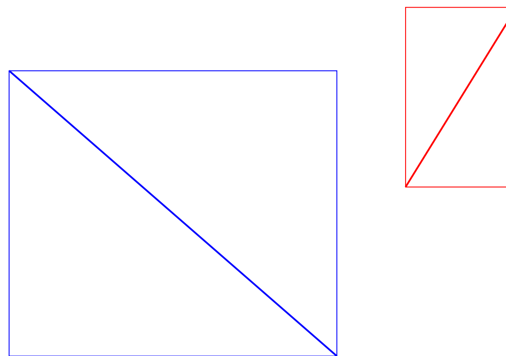
3. Check for intersection (most expensive)



Minimax Line Test

Let L_1 be the line segment between points (x_1, y_1) and (x_2, y_2)
Let L_2 be the line segment between points (x_3, y_3) and (x_4, y_4)

$$\left.\begin{array}{rcl} \mathtt{max}(x_1,x_2) & < & \mathtt{min}(x_3,x_4) \\ \mathtt{max}(x_3,x_4) & < & \mathtt{min}(x_1,x_2) \\ \mathtt{max}(y_1,y_2) & < & \mathtt{min}(y_3,y_4) \\ \mathtt{max}(y_3,y_4) & < & \mathtt{min}(y_1,y_2) \end{array}\right\} \text{if any are true, the lines do not intersect}$$



Check for Parallel Slopes

$$D = (x_3 - x_4)(y_1 - y_2) - (x_1 - x_2)(y_3 - y_4)$$

$$D = 0 \Rightarrow \text{equal slopes}$$

Actually, check $|D| < \varepsilon$

Check for Intersecting Edges

$$D = (x_3 - x_4)(y_1 - y_2) - (x_1 - x_2)(y_3 - y_4)$$

$$\begin{aligned} s &= [(x_3 - x_4)(y_1 - y_3) - (x_1 - x_3)(y_3 - y_4)]/D \\ t &= [(x_1 - x_2)(y_1 - y_3) - (x_1 - x_3)(y_1 - y_2)]/D \end{aligned}$$

If $0 < (s,t) < 1$, the lines intersect at (x,y), where

$$\begin{aligned} x &= x_1 + s(x_2 - x_1) \\ y &= y_1 + s(y_2 - y_1) \end{aligned}$$

Check for Containment

Do minimax test with center of one triangle and containing rectangle of the other

Hidden Surface Removal

Also called *visibile surface determination*

Two approaches:

- Image precision algorithm
- Object precision algorithm

Image Precision Algorithm

```
for ( each pixel p in the image ) {

        Determine the object closest to the viewer
                that is pierced by a projector through p;

        Draw p with the appropriate color;

}
```

Object Precision Algorithm

for (each object obj in the scene) $\{$

Determine those parts of obj whose
view is unobstructed by other
parts of obj or any other object

Draw those parts of obj with the appropriate
color;

$\}$

Comparing the Approaches

- Image precision
  - Speed
  - Initially for raster displays
  - Used by ray tracers
- Object precision
  - Accuracy
  - Initially for vector displays

Comparing the Approaches

- Image precision
  - For n objects and p pixels, the effort is $O(np)$
  - $p > 1,000,000$ for high resolution displays
- Object precision
  - For n objects, effort is $O(n^2)$
  - Better for $n < p$? No; each step is more complex

Coherence

- Visible surface determination algorithms can exploit *coherence*
- Parts of a view's environment (or portion of that environment) often exhibit local similarities
- Some properties vary smoothly from one part to another

Kinds of Coherence

- Object coherence
- Face coherence
- Edge coherence
- Implied edge coherence
- Scan line coherence
- Area coherence
- Depth coherence
- Frame coherence

Object Coherence

For two separate objects comparisons may need to be done only between the two objects as a whole, not between their components (faces, edges)

If all of object A's parts are farther away than object B's parts, there is no need to compare B's faces to A's faces

Face Coherence

Surface properties often vary smoothly across a face, so computations for one part of a face may be modified incrementally to apply to adjacent parts

Edge Coherence

An edge may change its visibility only where it crosses behind a visible edge or penetrates a visible surface

Implied Edge Coherence

The line of intersection between two penetrating planes can be determined from two points of intersection

This line of intersection is an implied edge

Scan Line Coherence

The set of visible object spans determined from one scan line differs little from the set on the previous scan line

Area Coherence

A group of adjacent pixels is often covered by the same visible face

Span coherence refers to area coherence on a given scan line

Depth Coherence

Adjacent parts of the same surface are usually close in depth

Different surfaces at the same screen location usually have a greater depth difference

Frame Coherence

Scenes from two successive points in time are likely to be quite similar, despite small changes in object's positions or change in viewpoint

Some calculations made for one frame may be able to be reused or the results slightly modified for the next frame

z-buffer Algorithm

- An image-precision algorithm
- Simple to implement in software and hardware
- Requires two buffers
  - Frame buffer for color values of each pixel
  - z-buffer of the same size holding the depth of the front-most object at that pixel

z-buffer Algorithm

```
zBuffer() {
    for ( y = 0; y < MAX_Y; y++ ) {
        for ( x = 0; x < MAX_X; x++ ) {
            refreshBuffer[x][y] = BACKGROUND_COLOR;
            zBuffer[x][y] = ∞;
        }
    }
    for ( each polygon p ) {
        for ( each pixel (x, y) in p's projection ) {
            p_z = p's z-value at pixel (x, y);
            if ( p_z < zBuffer[x][y] ) {
                zBuffer[x][y] = p_z;
                refreshBuffer[x][y] = p's color at (x, y);
            }
        }
    }
}
```

z-buffer Algorithm

- No need to do any sorting
- No object-object comparisons needed
- Each polygon is scan converted
  - Color at a screen pixel is stored in the frame buffer (if z value is less)
  - Depth at that position is stored in the z-buffer (if z value is less)

z-buffer Algorithm

- Rearranging the plane equation gives z as a function of x and y:

$$z = \frac{-D - Ax - By}{C}$$

- This gives the depth at (x, y)

z-buffer Algorithm

- The next z value in a scan line can be computed incrementally:

$$f_z(x,y) = \frac{-D - Ax - By}{C} = z_1$$

$$f_z(x + \Delta x, y) = z_1 - \frac{A}{C}(\Delta x)$$

- Given $f_z(x,y)$, only one subtraction is required to compute $f_z(x+1,y)$

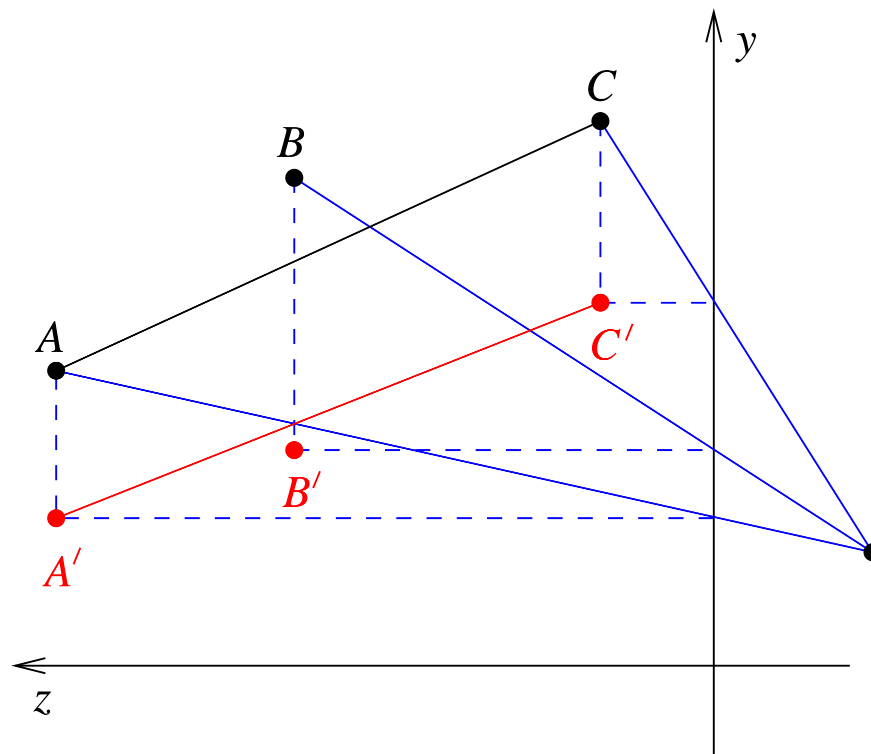
Scan Line Algorithms

- z-buffer need not be so large
- Use a scan line buffer
  - z-buffer is one scan line big
  - Sort polygons by y value
  - Idea similar to scan line polygon fill

Perspective Depth Transformation

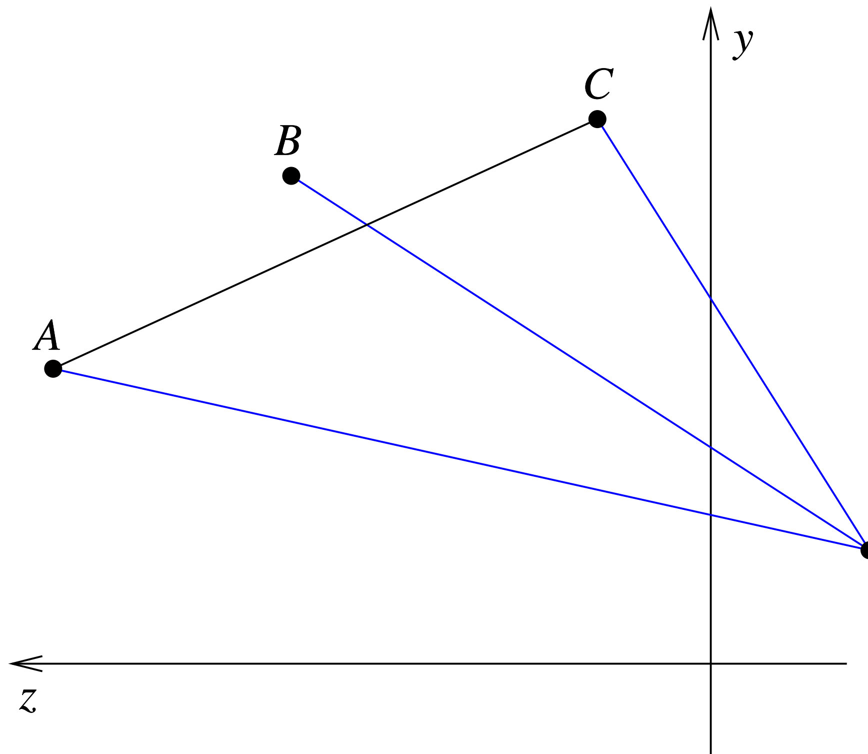
- When depth is not important, the z coordinate can be dropped when projecting the 3D point in 2D
- Simply dropping the z coordinate results, however, in a non-linear transformation that causes some depth computations to be incorrect

Bad Depth Calculation



$$
\begin{aligned}
x_p &= \frac{d \cdot x}{d + z} \\
y_p &= \frac{d \cdot y}{d + z} \\
z_p &= z
\end{aligned}
$$

Correct Depth Calculation



$$x_p = \frac{d \cdot x}{d + z}$$

$$y_p = \frac{d \cdot y}{d + z}$$

$$z_p = \frac{d \cdot z}{d + z}$$

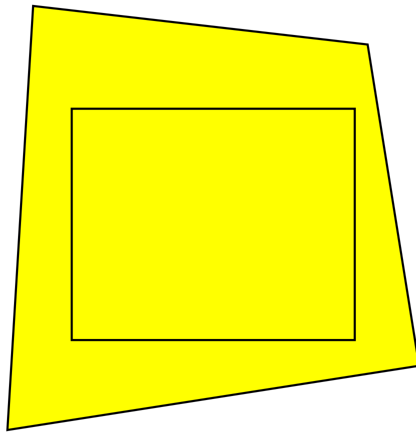
Screen Subdivision Methods

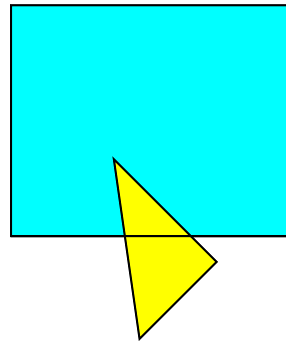
- Use a divide-and-conquer strategy
  - Examine an area of the screen
  - If it is easy to decide which polygons are visible in that area, then display them;
  - Otherwise, subdivide that area into smaller areas and recursively apply the decision logic to each region
- As the regions become smaller it becomes easier to decide

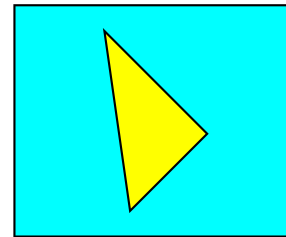
Warnock's Algorithm

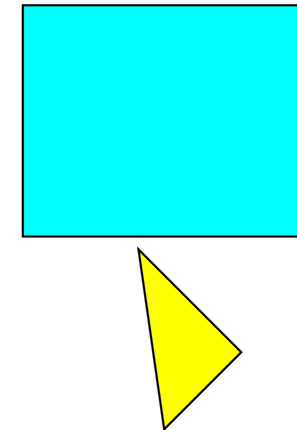
- An area subdivision method that divides the screen into four regions
- Four relations of polygon projections are considered:
  - Surrounding
  - Intersecting
  - Contained
  - Disjoint

Warnock's Algorithm Region Relations



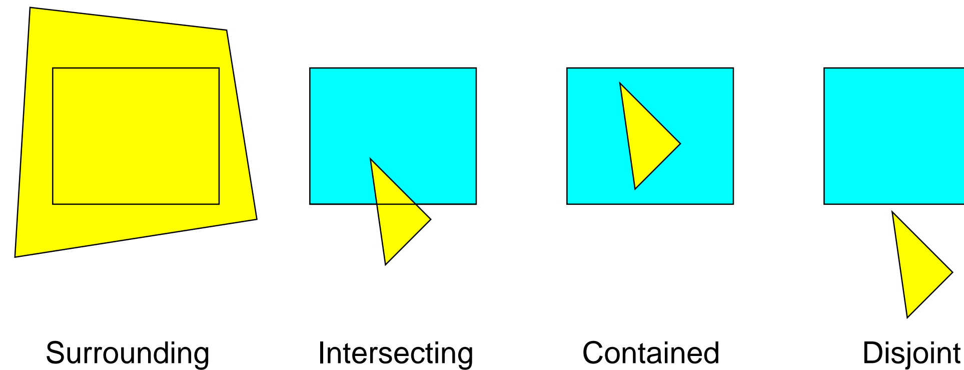






Surrounding Intersecting Contained Disjoint

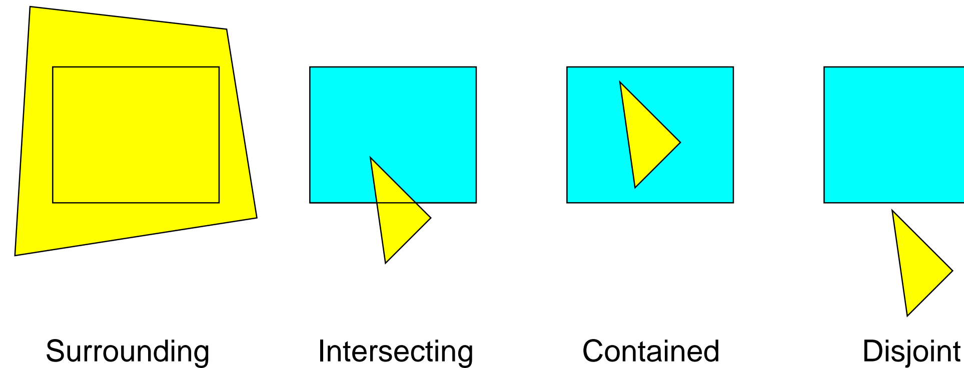
Simple Cases



No subdivision required

- All polygons disjoint from the area—simply draw background area
- Only one intersecting or contained polygon—fill area with background, then fill polygon or portion thereof

Simple Cases (cont.)



- Single surrounding polygon but no intersecting or contained polygons—fill area with pixel color of surrounding polygon
- More than one polygon is surrounding, intersecting, or contained in the area, but one surrounding polygon is in front of all the others—if the four corner z coordinates are closer to the viewport than are any of the other polygons, then fill the area with that polygon's color

Subdivision

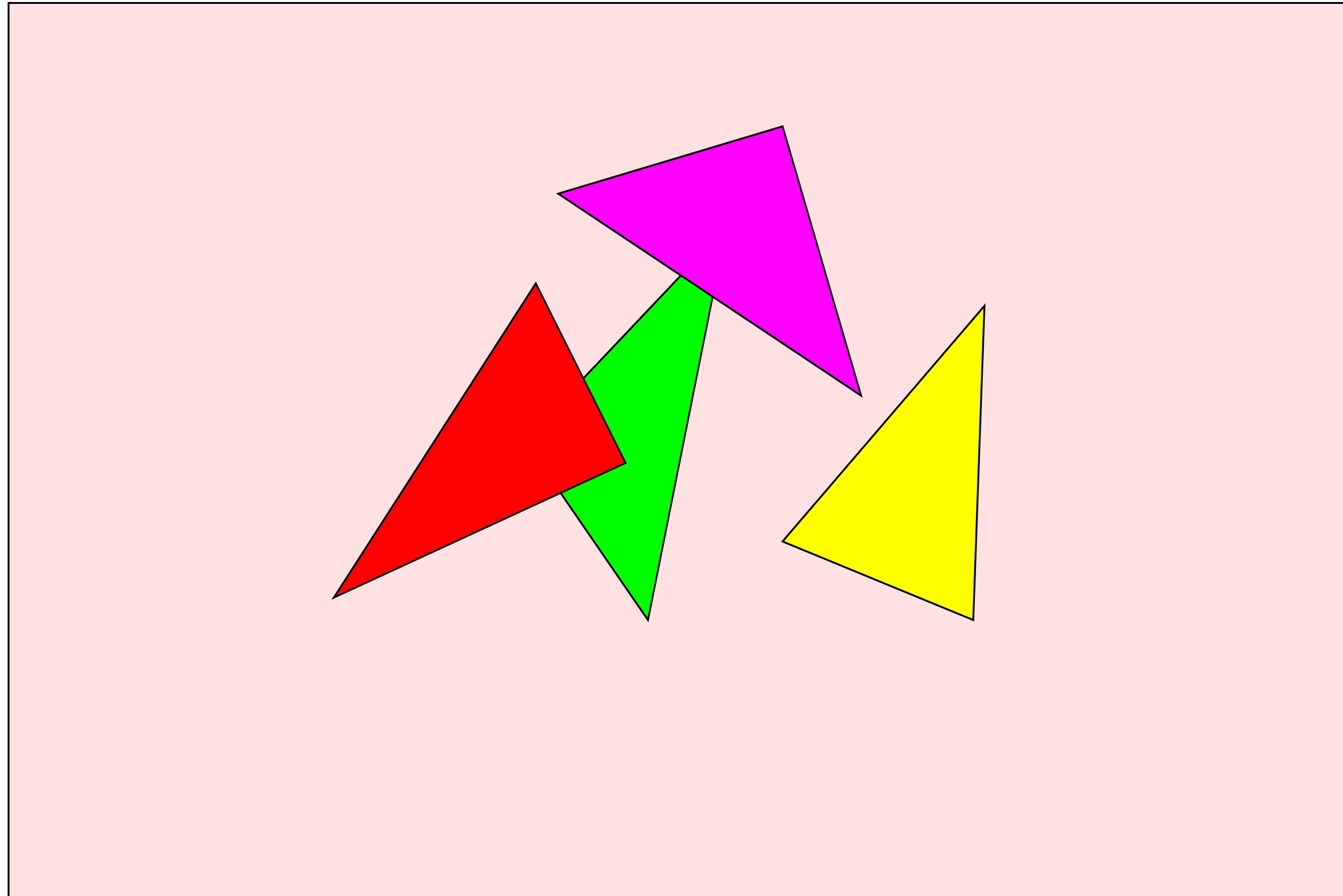
- After subdivision only contained and intersecting polygons need be re-examined (surrounding and disjoint polygons remain so in the subdivided region)
- As the area becomes smaller, fewer polygons will overlap each area, and then decision will eventually become possible

Area Coherence

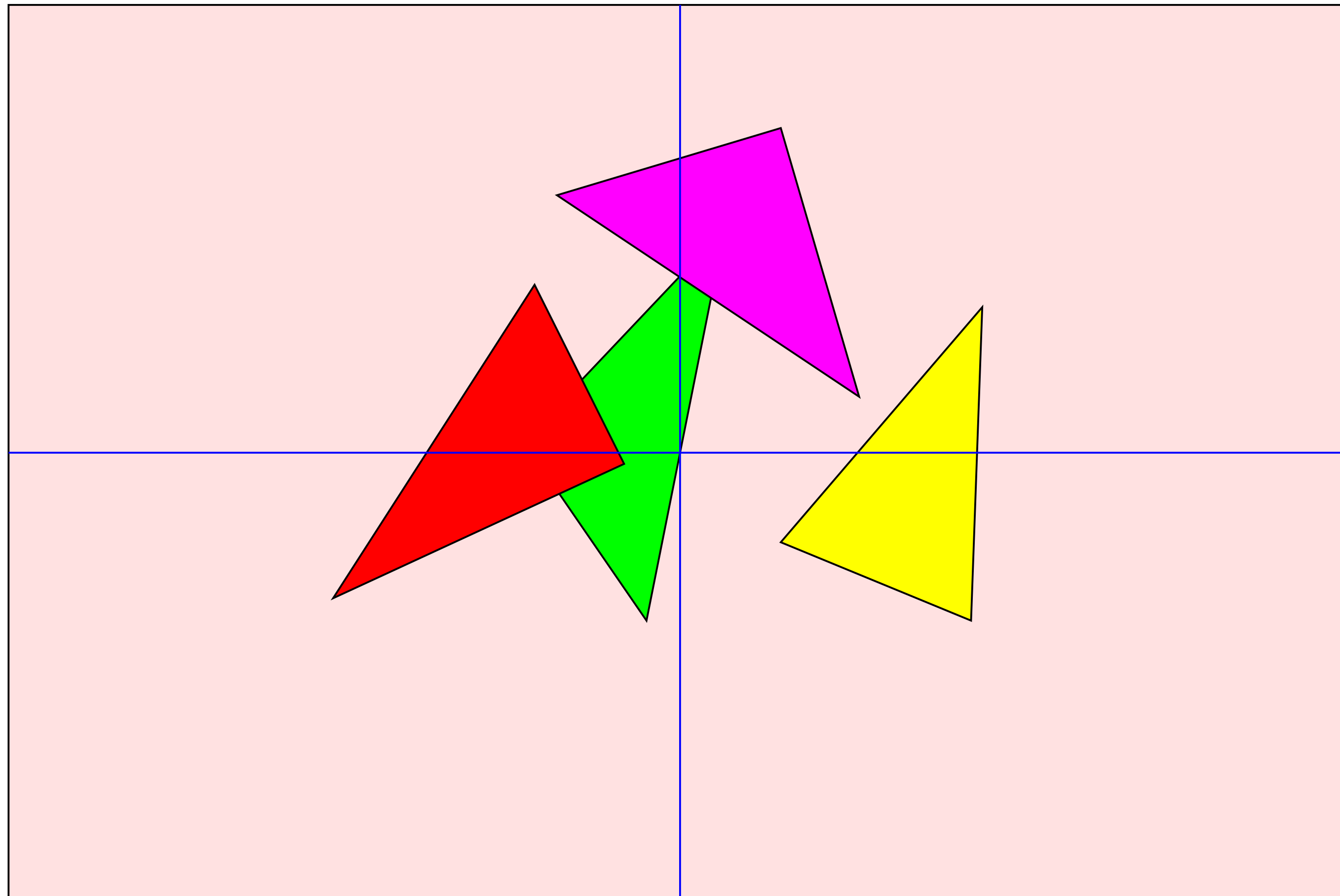
- Warnock's algorithm exploits area coherence—the tendency for small areas of an image to be contained in at most a single polygon

- Subdivisions end when the limits of the resolution of the display device has been reached
  - On a 1024×1024 display at most 10 subdivisions are necessary

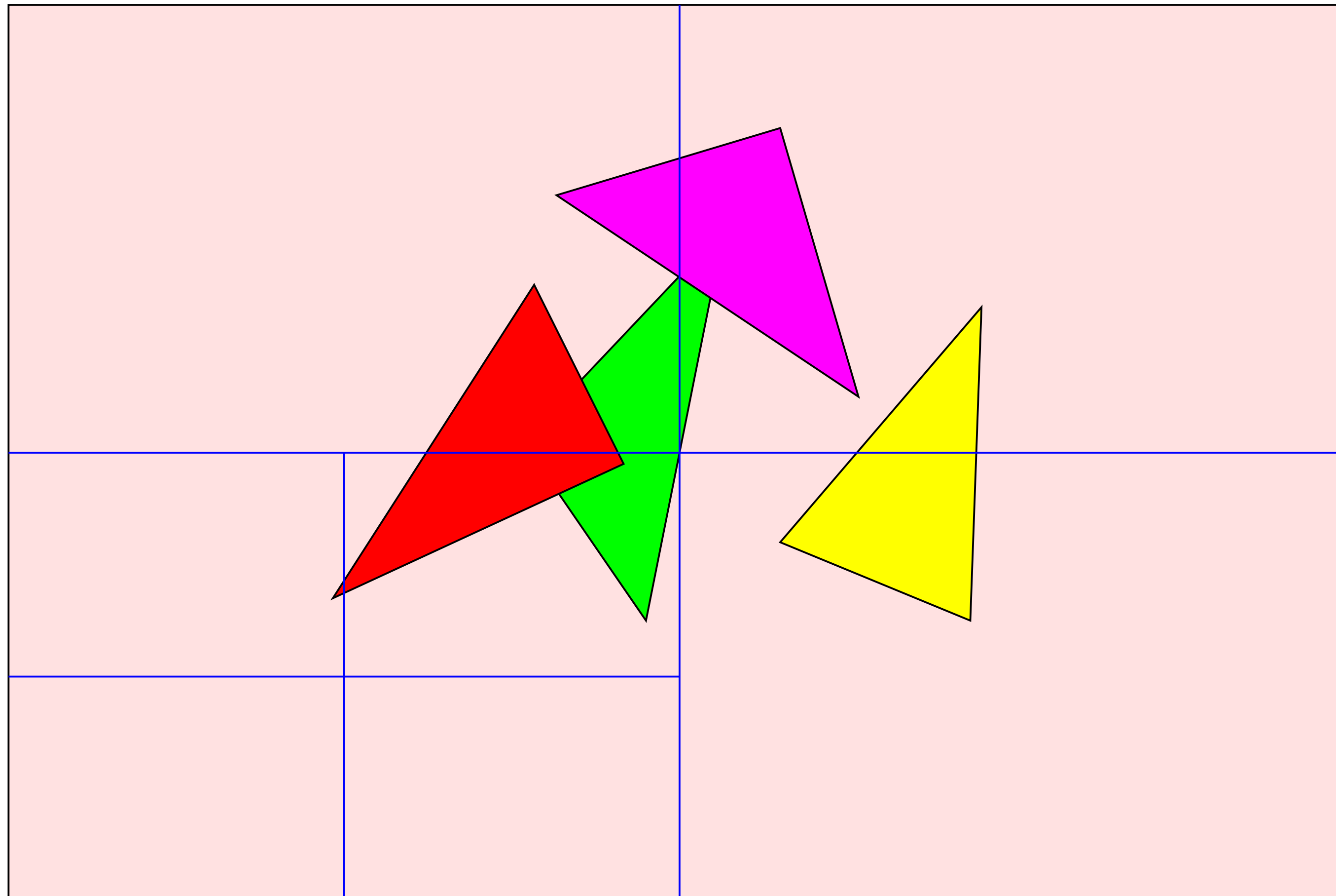
Example



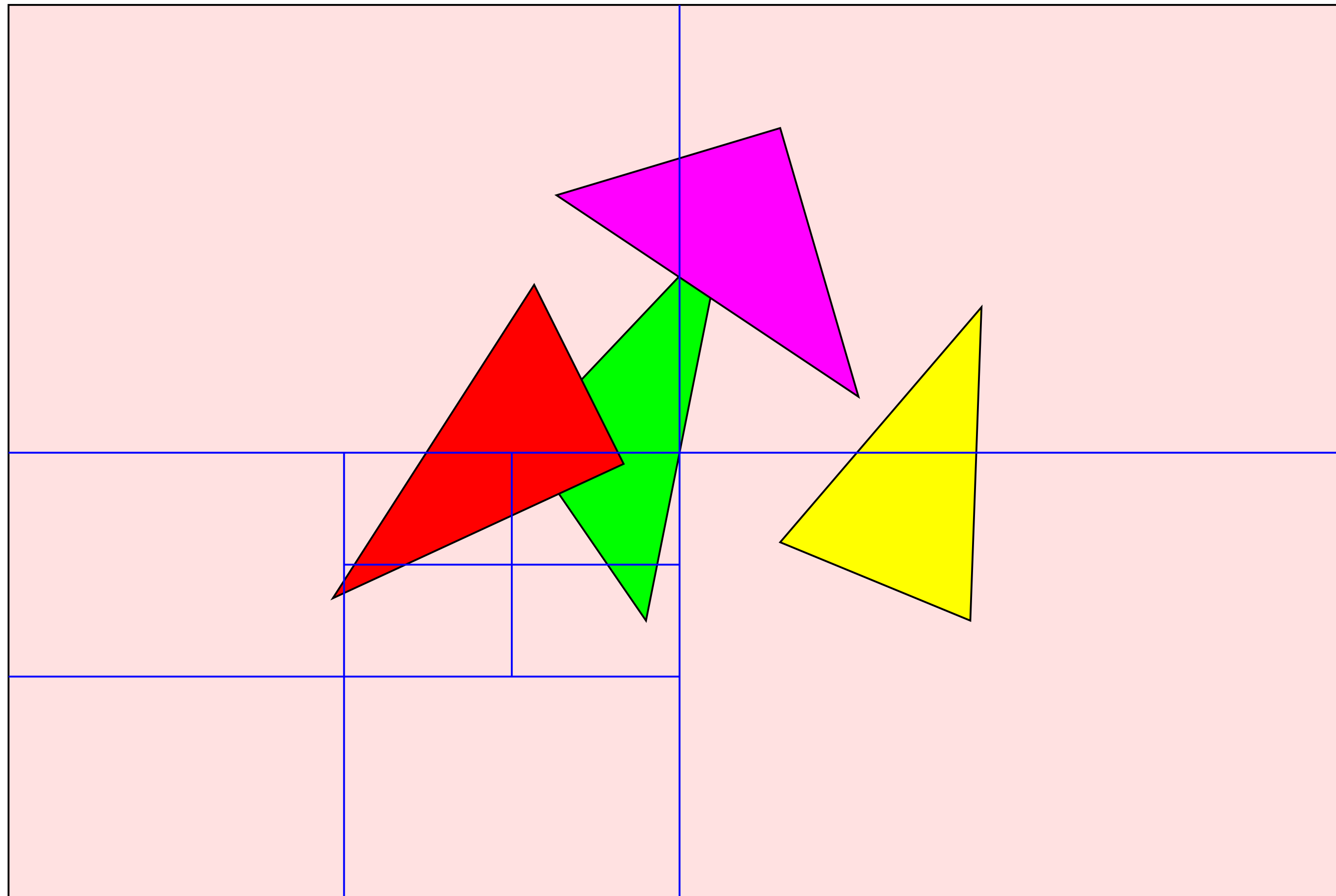
Example



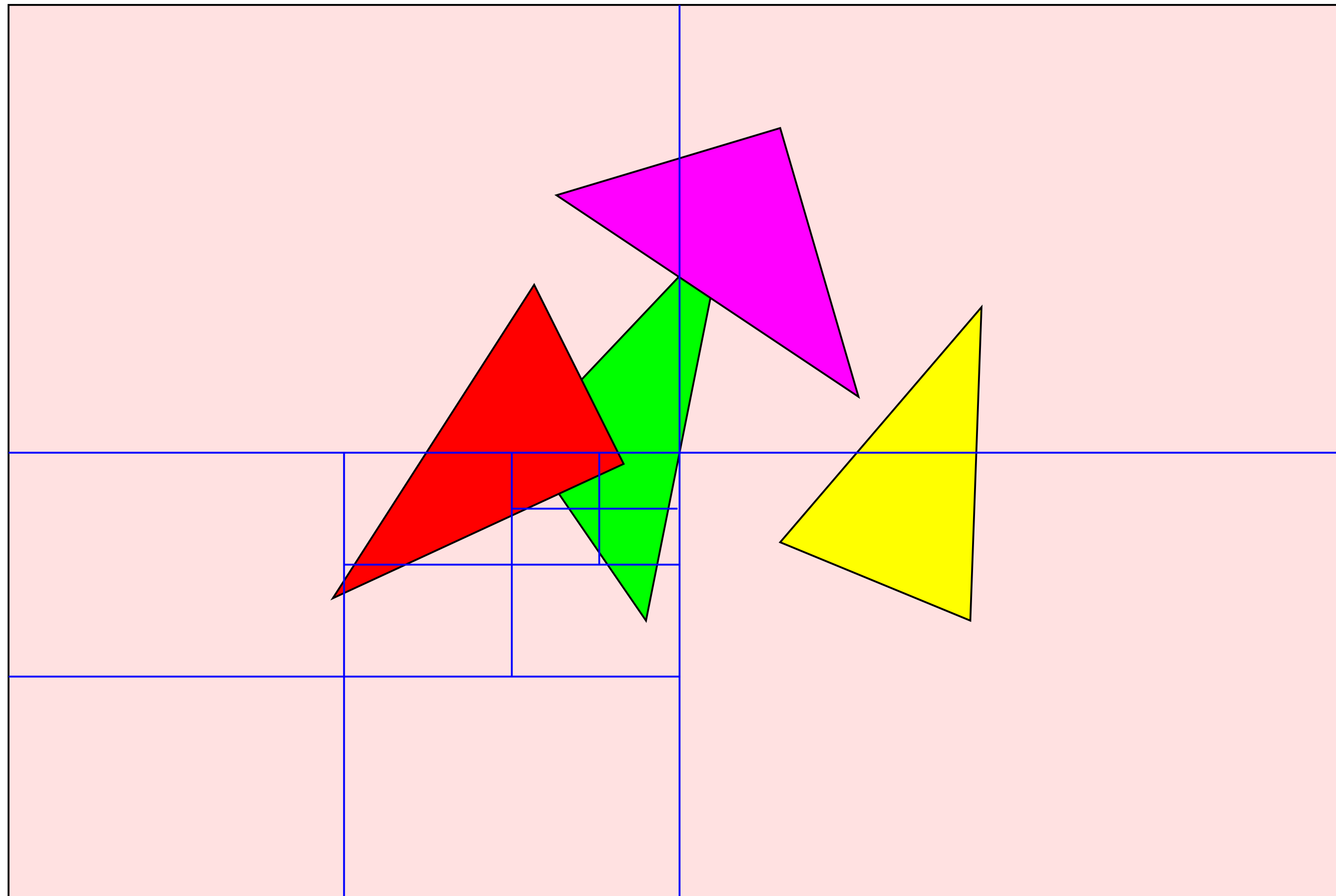
Example



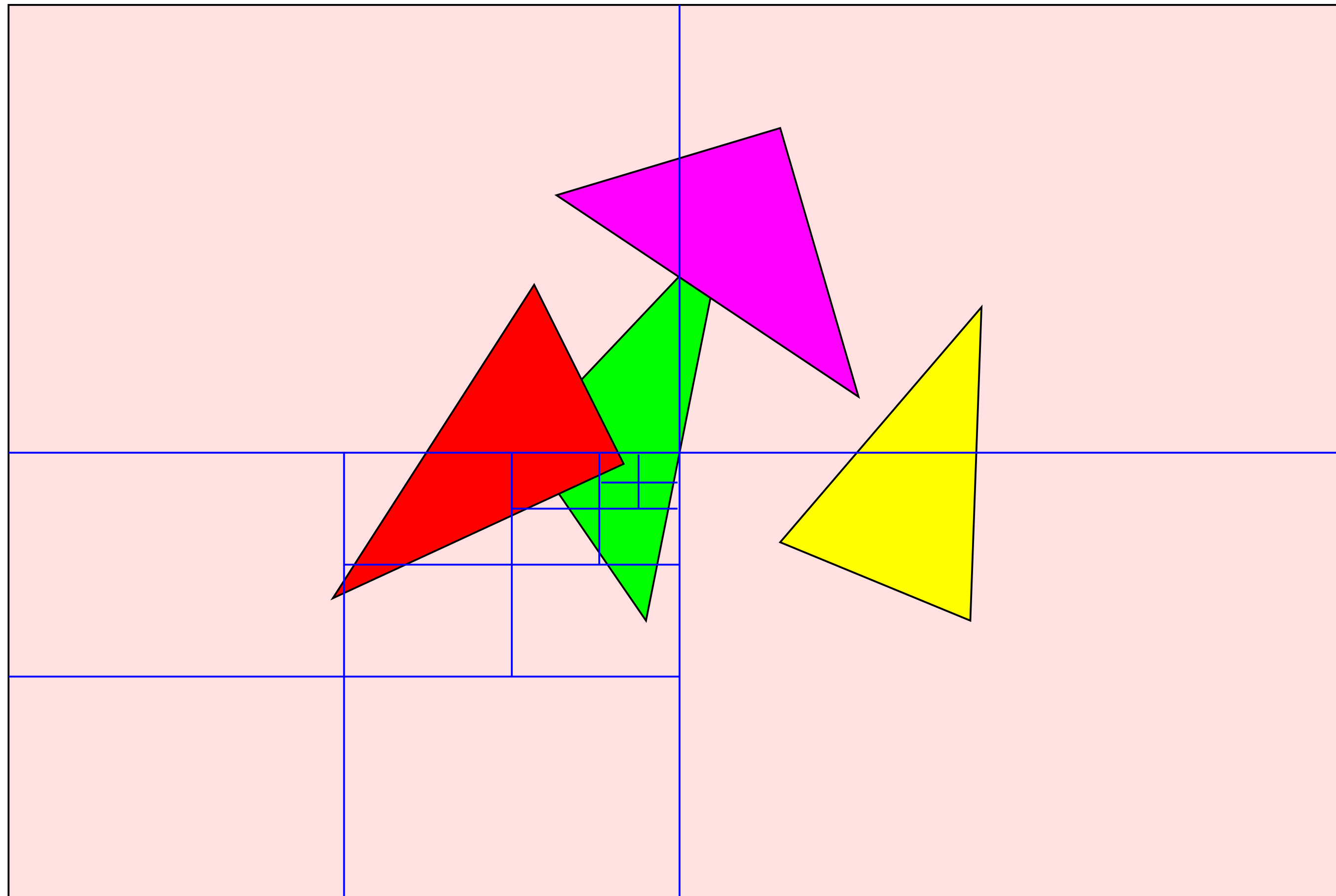
Example



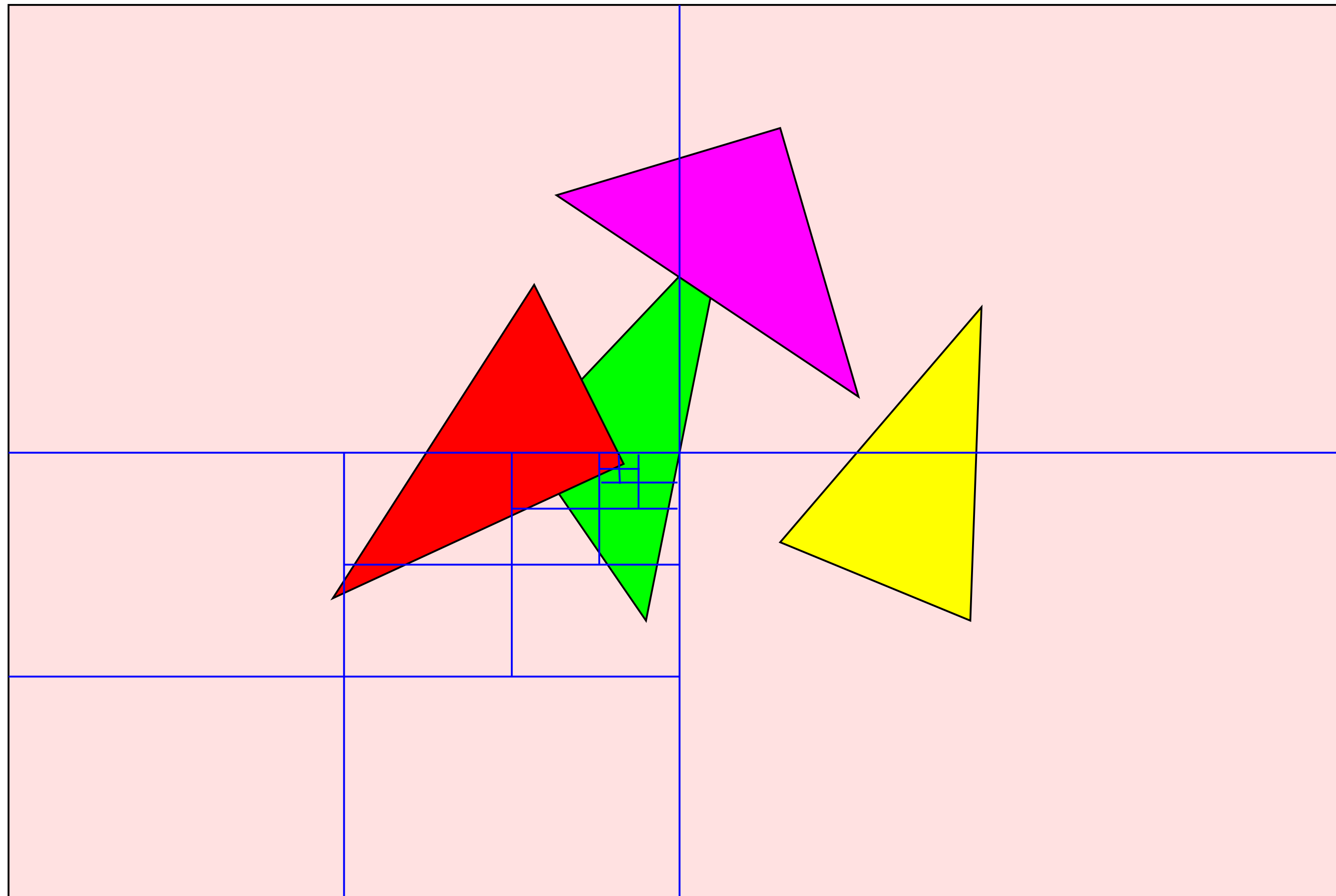
Example



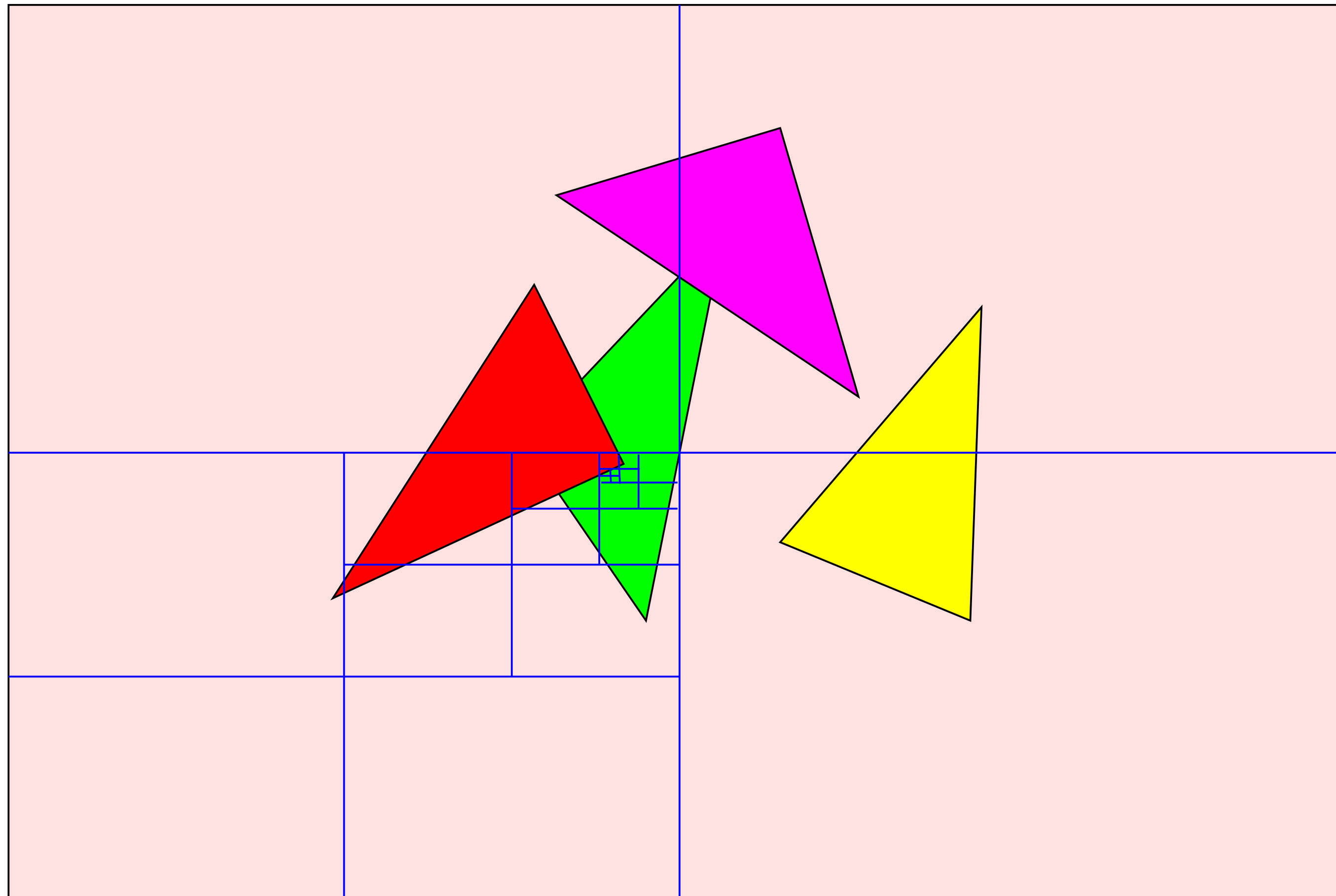
Example



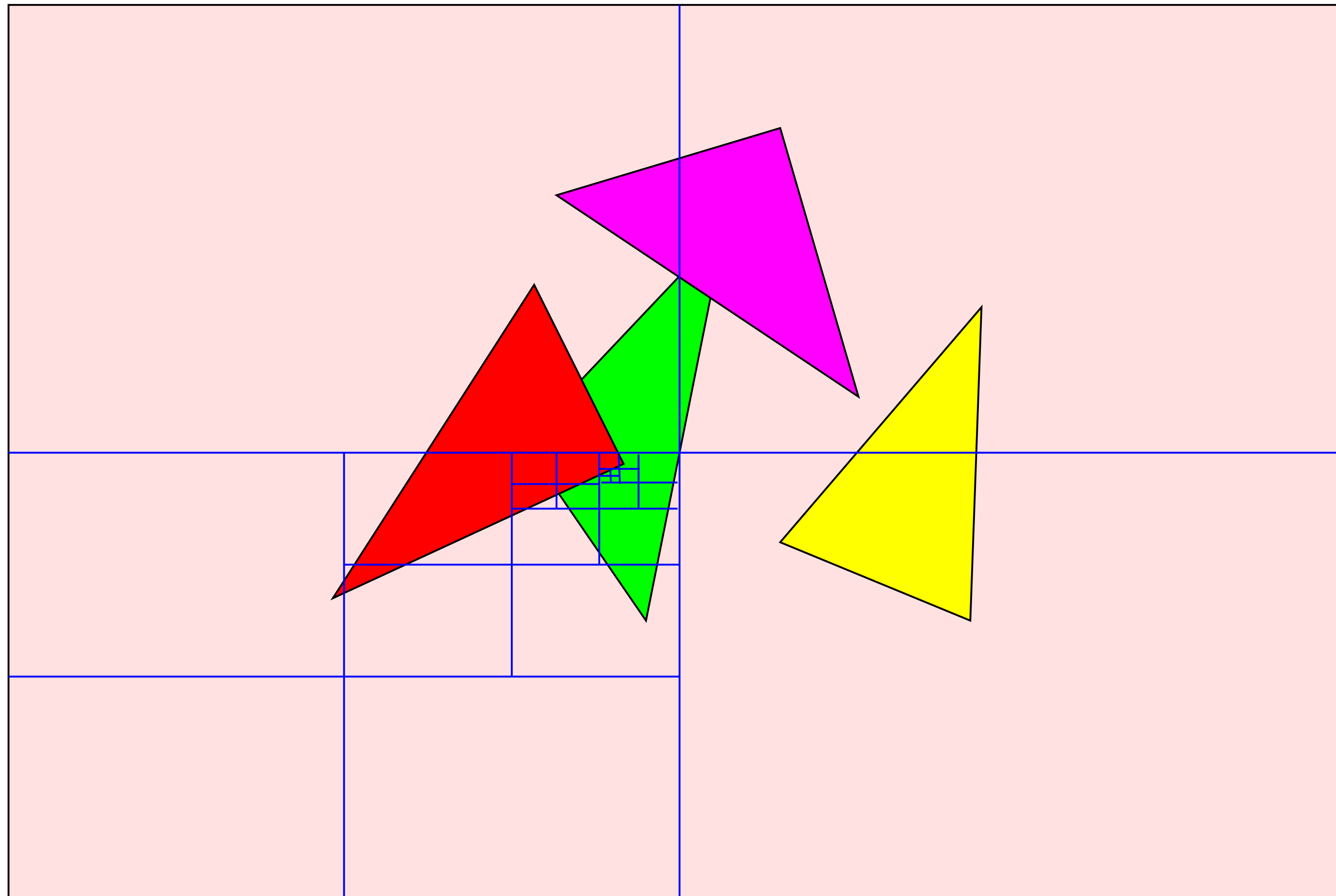
Example



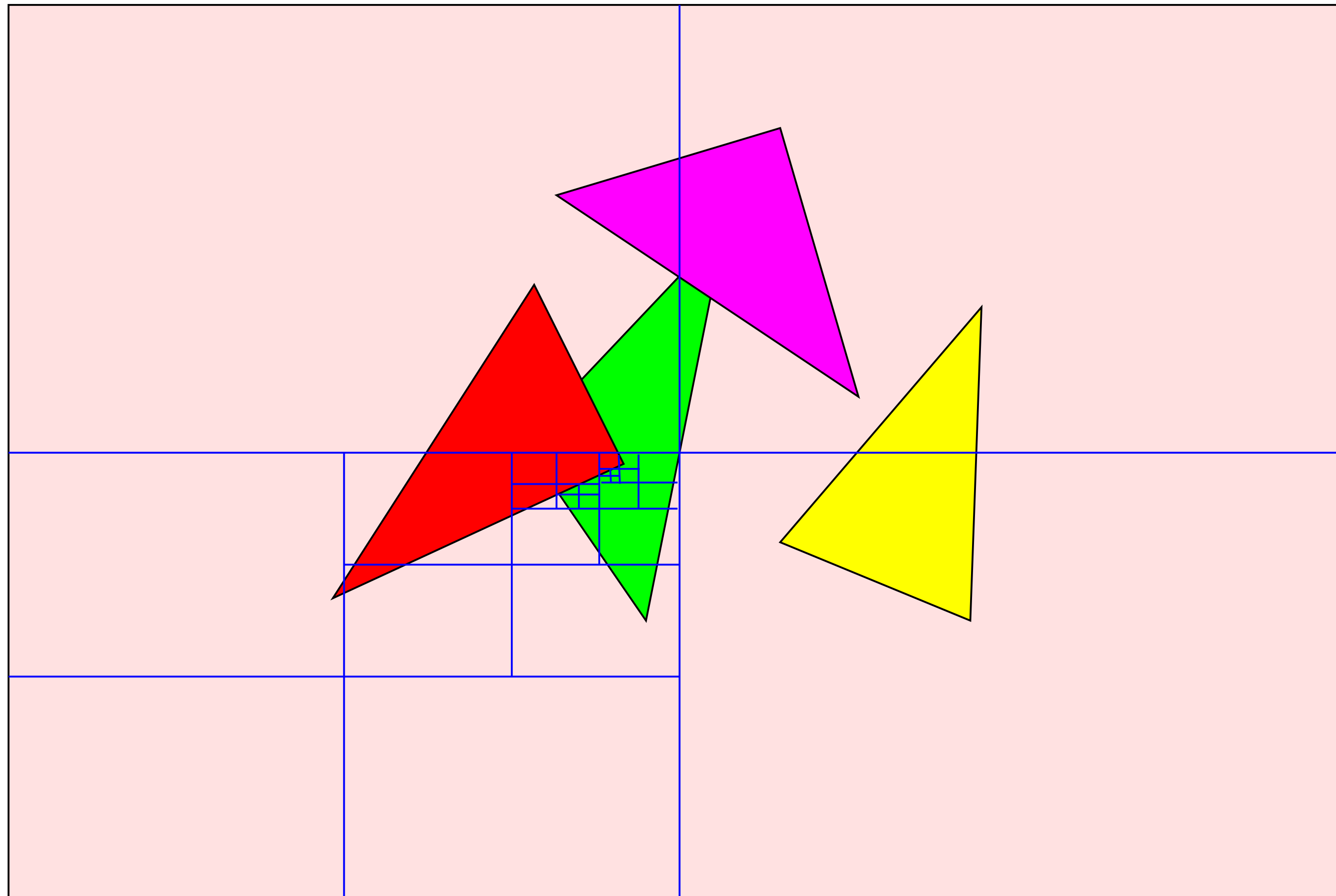
Example



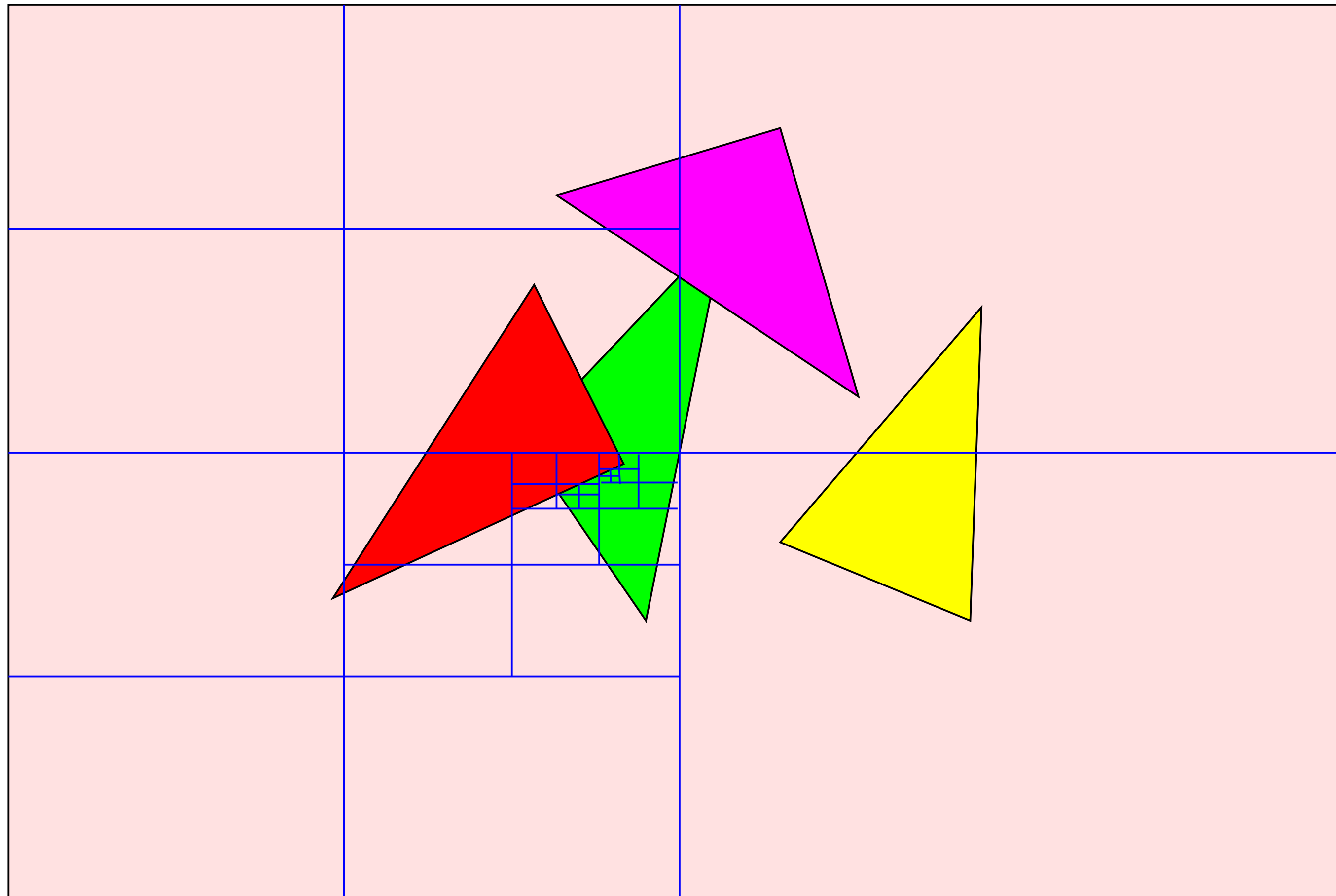
Example



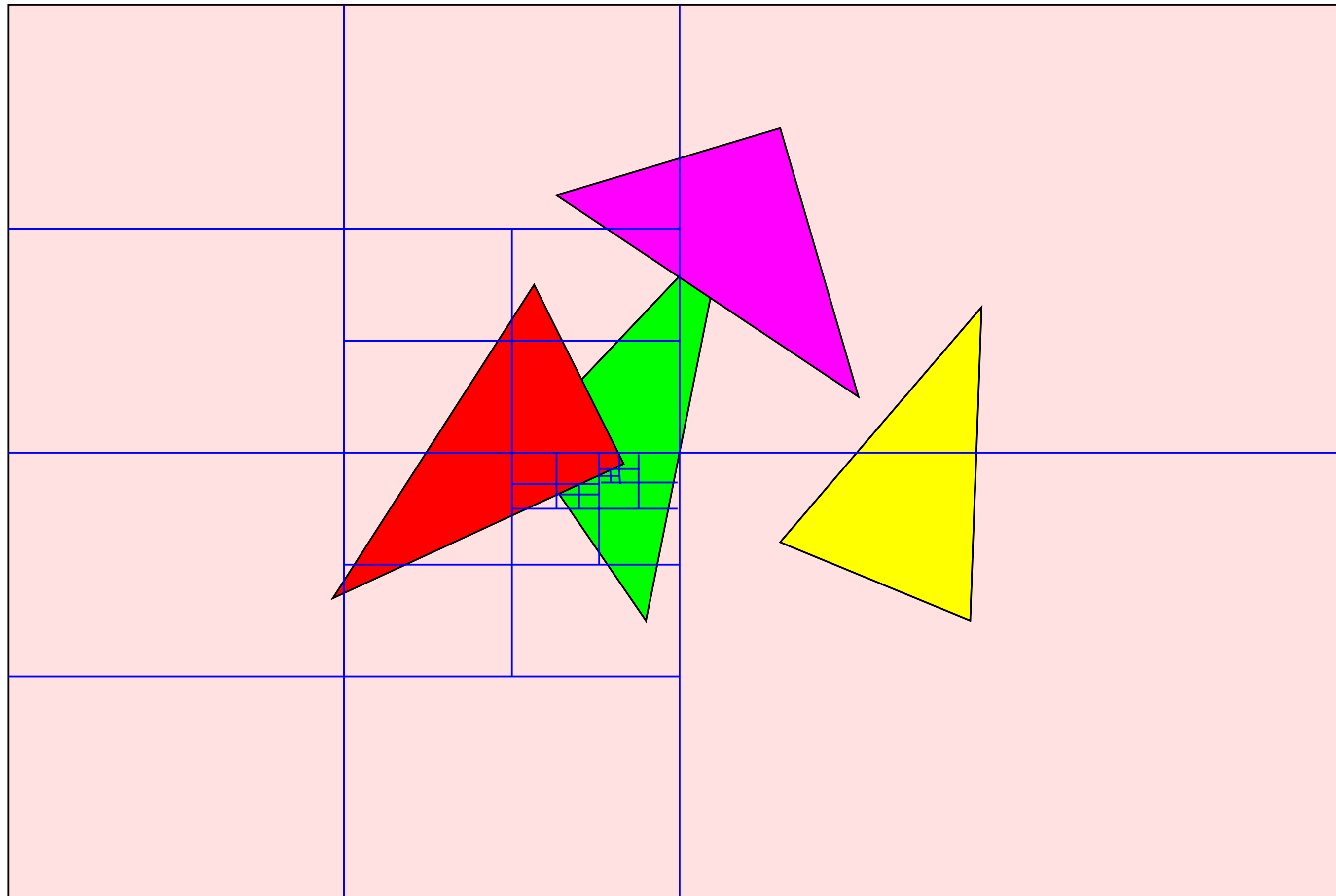
Example



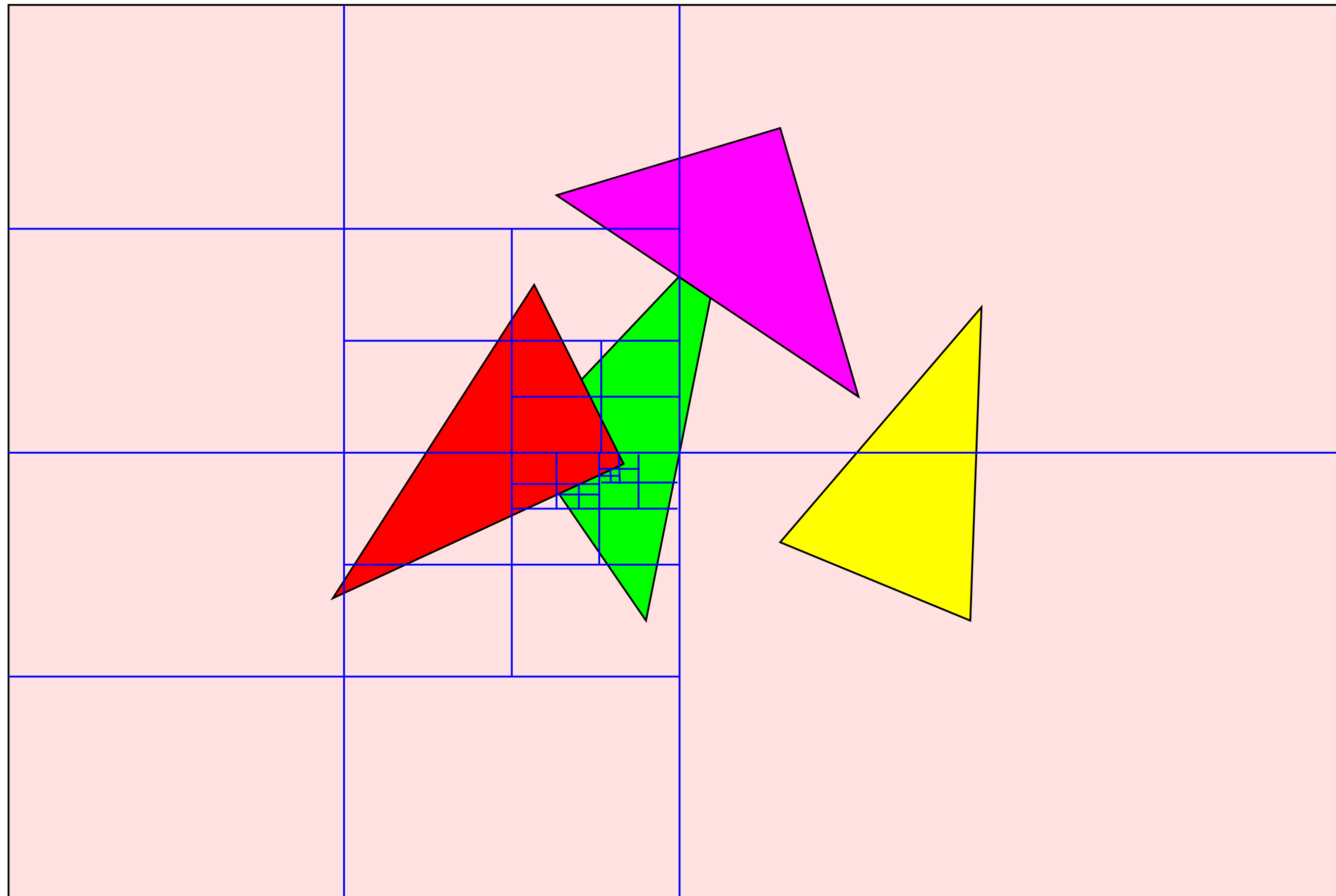
Example



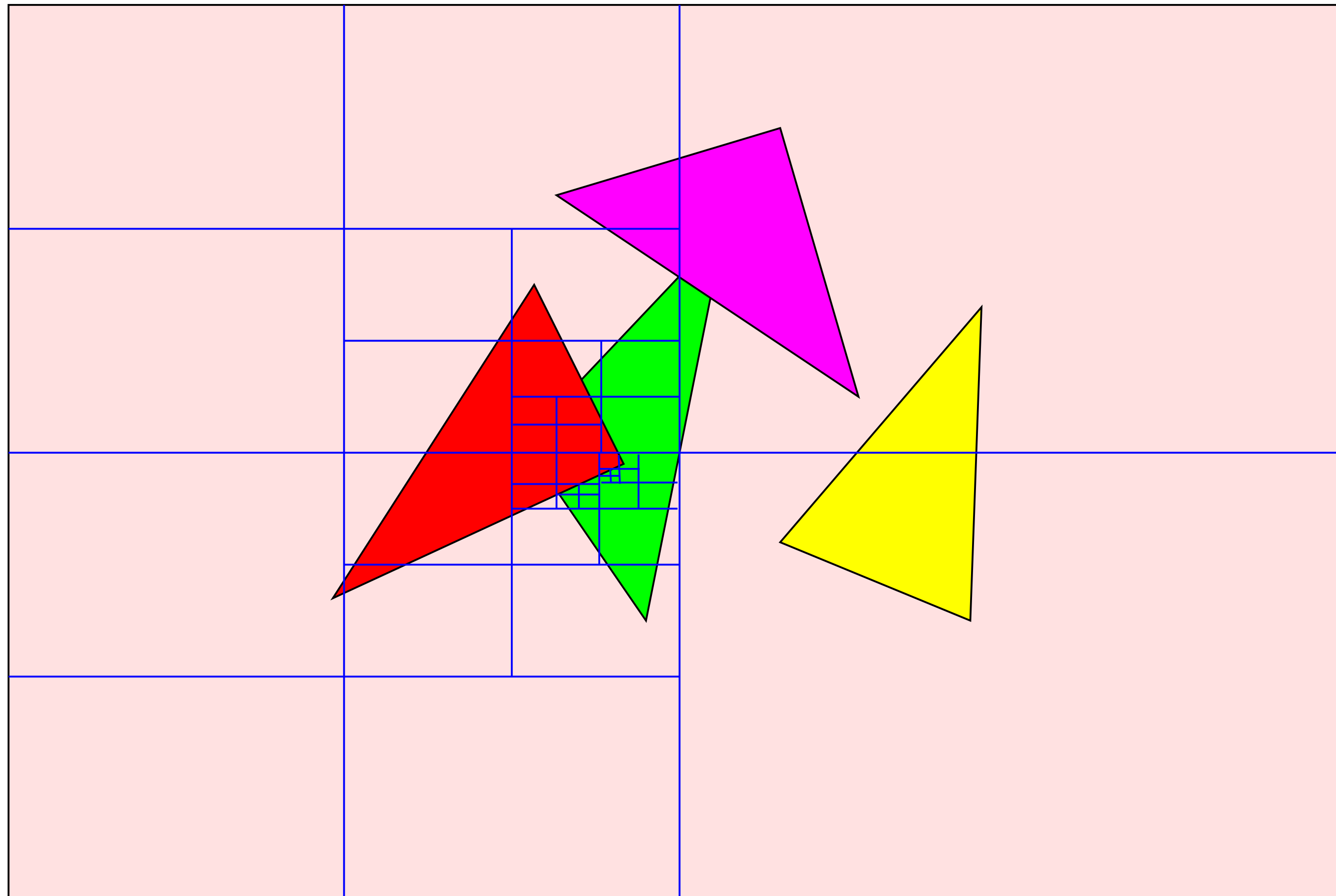
Example



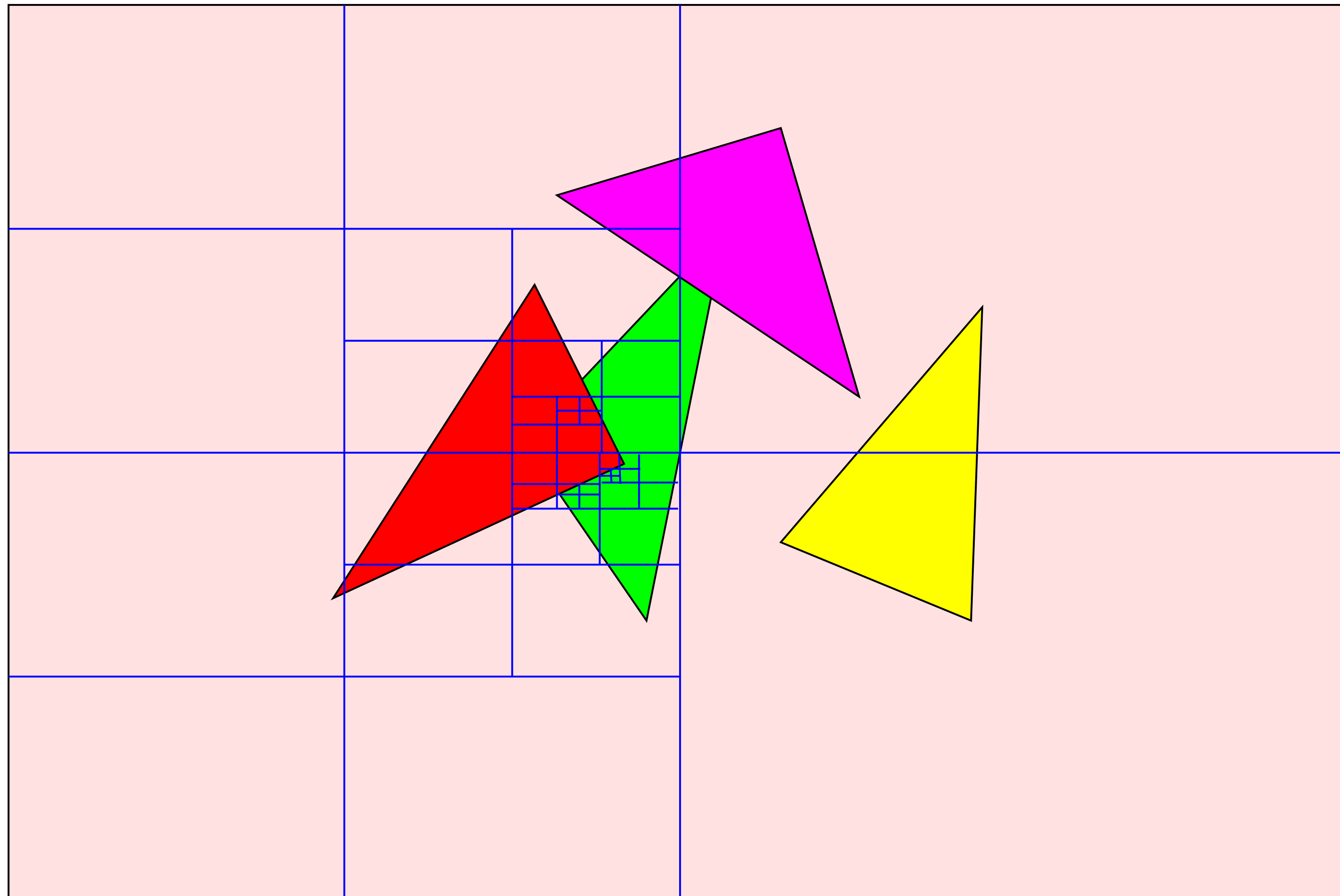
Example



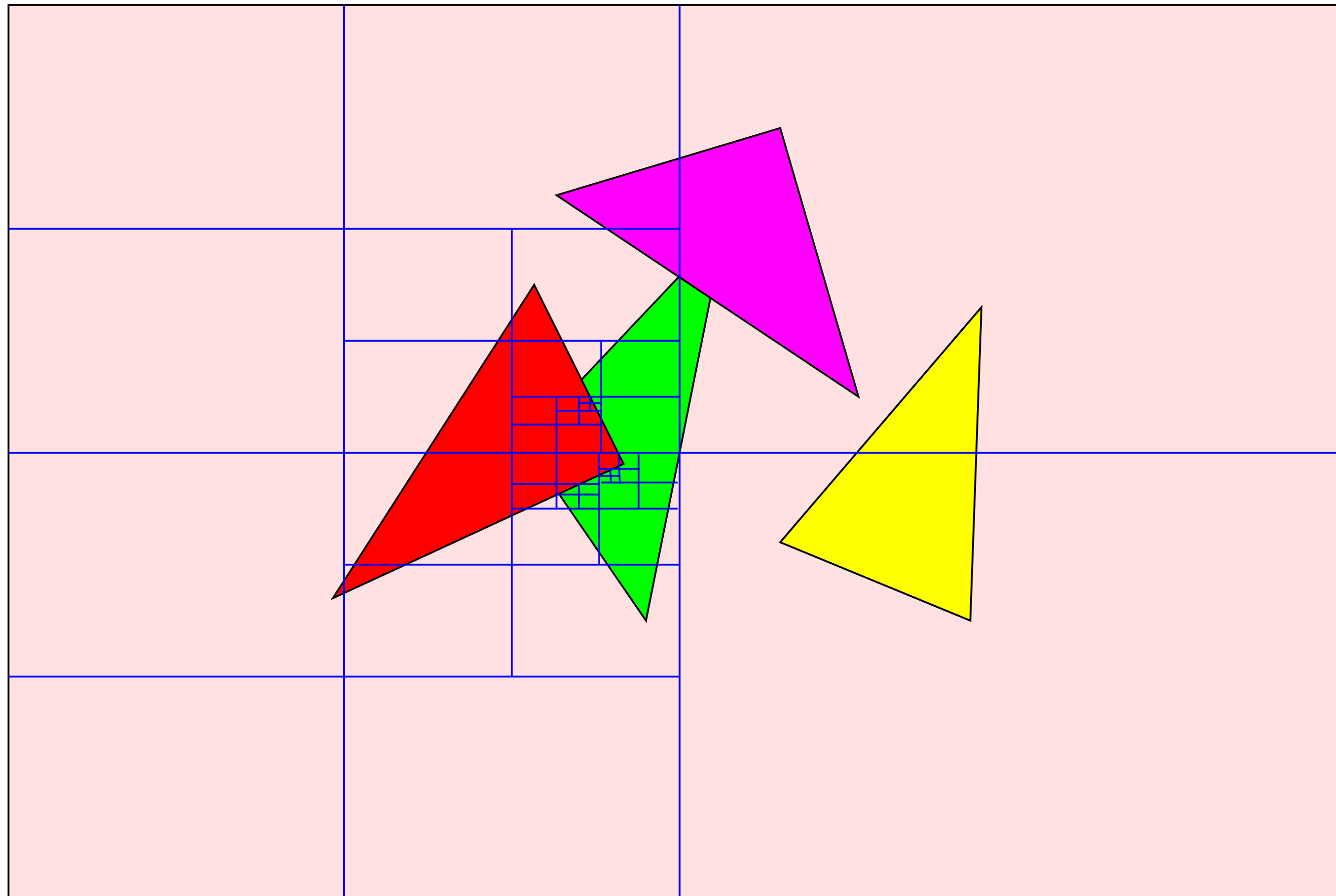
Example



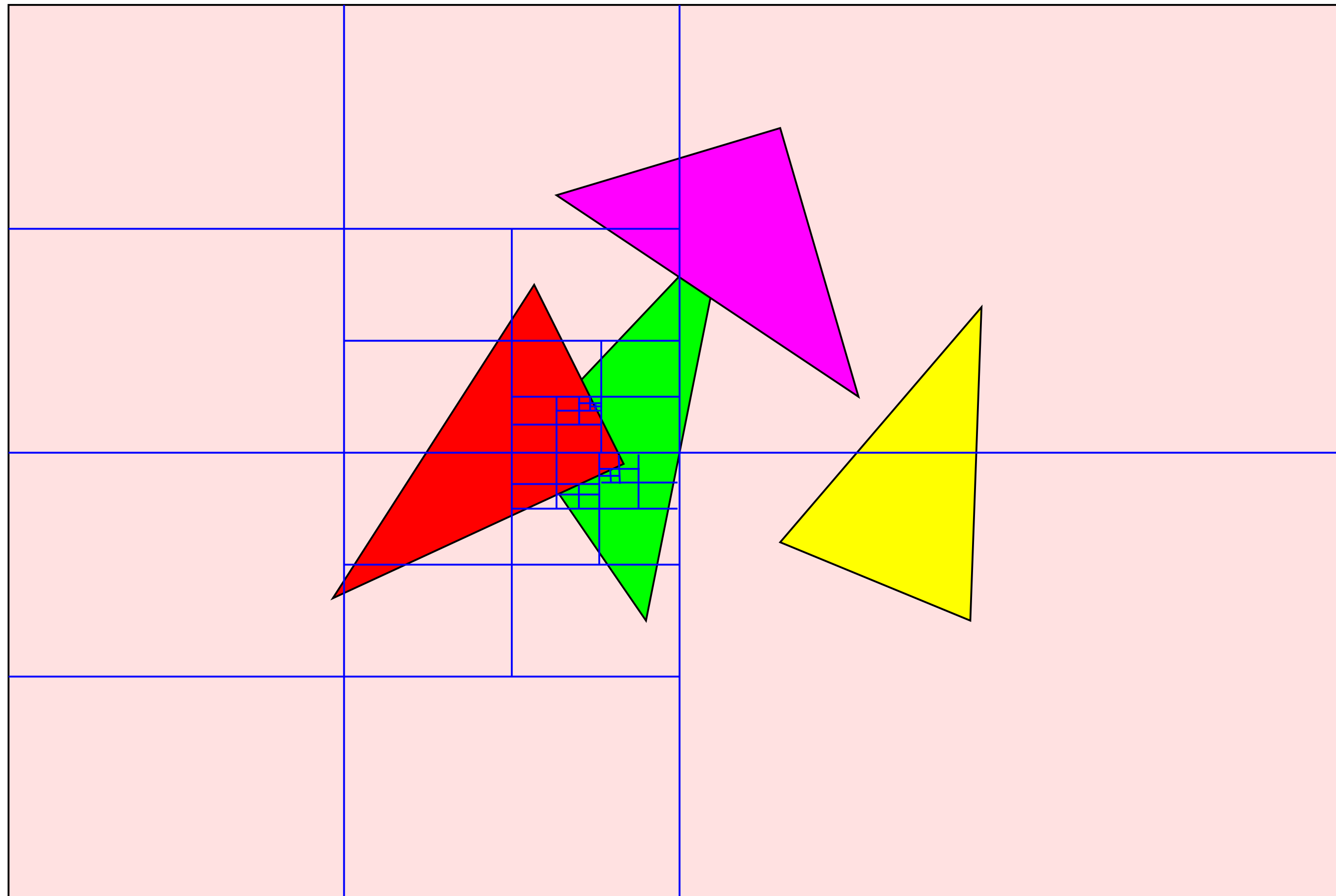
Example



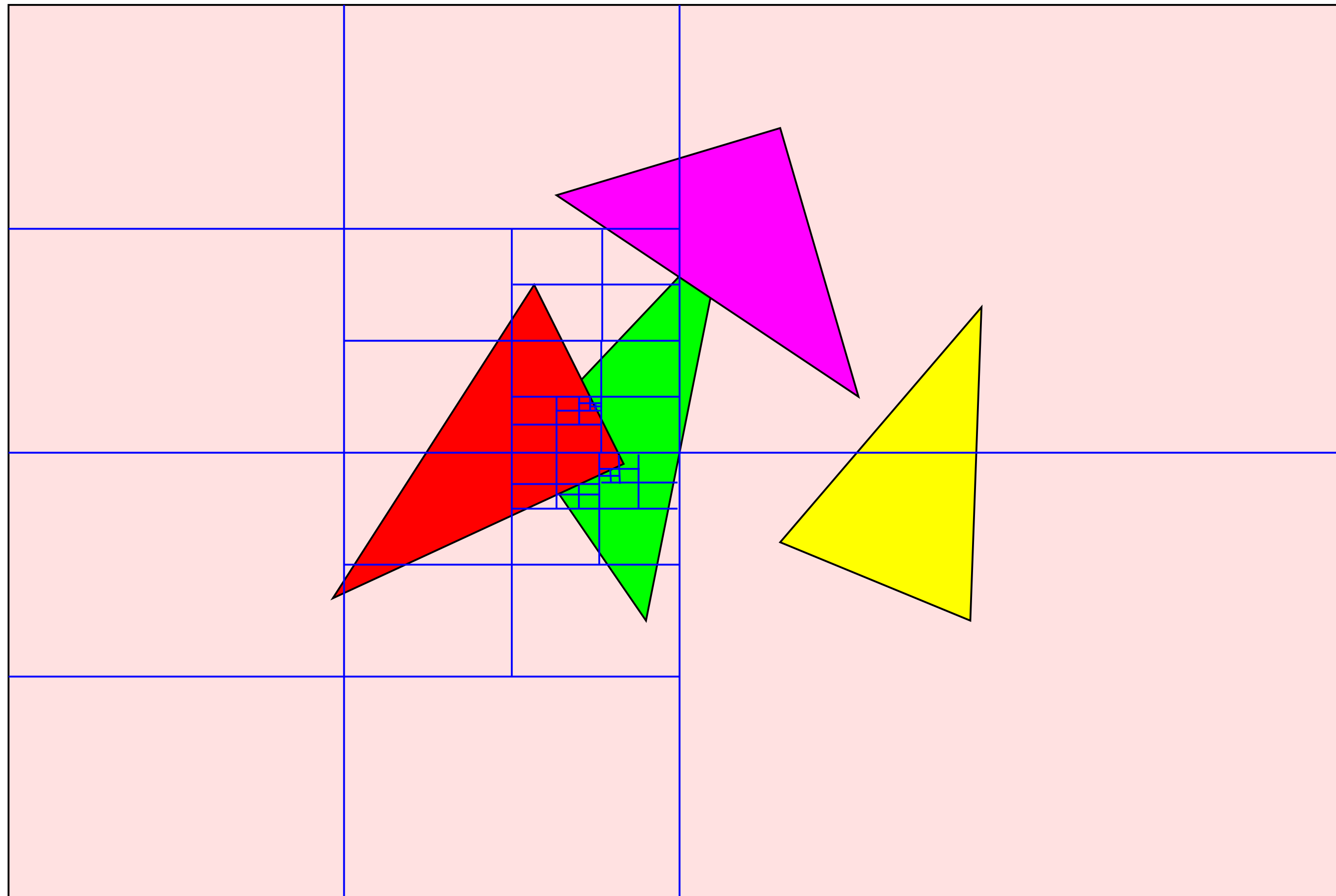
Example



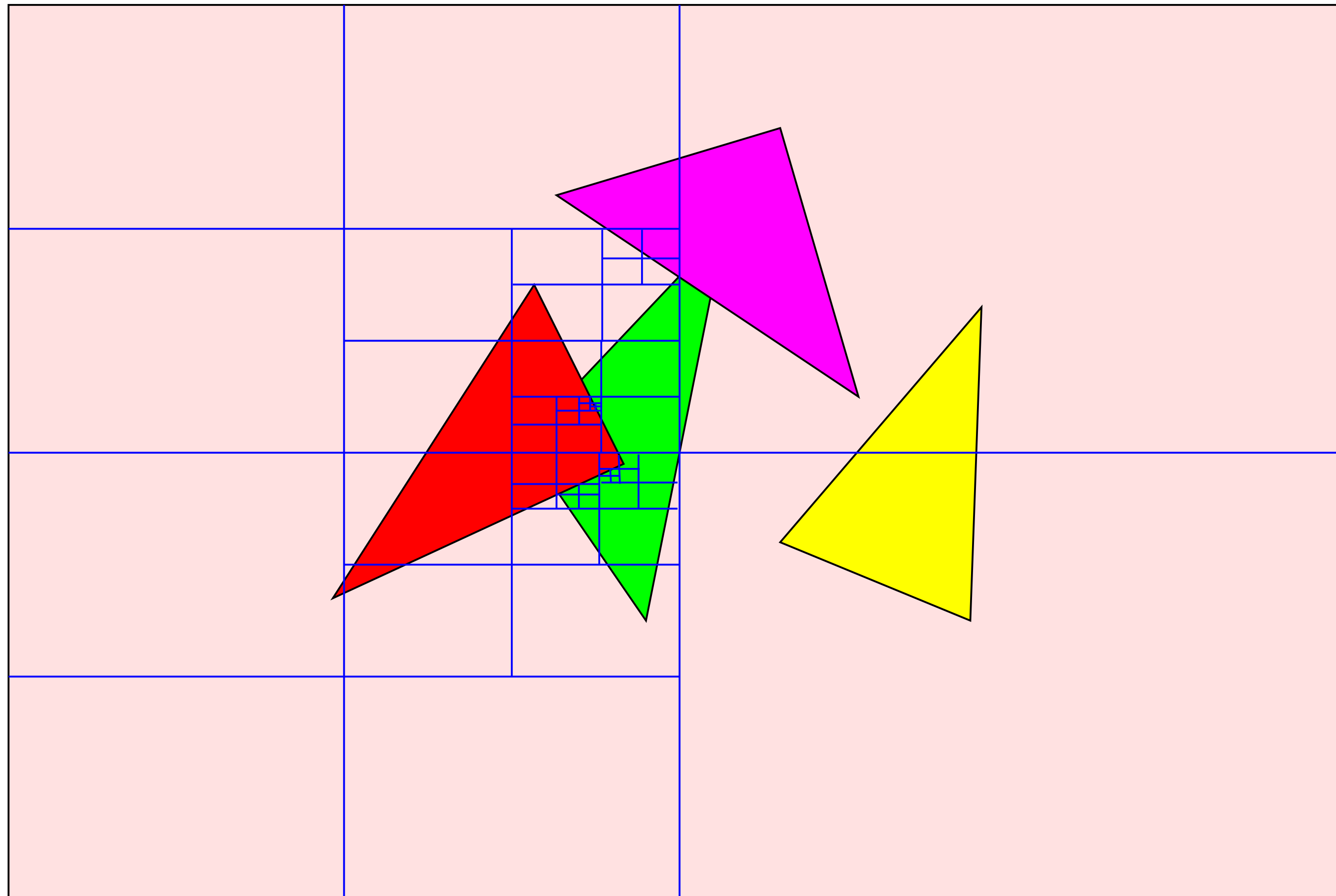
Example



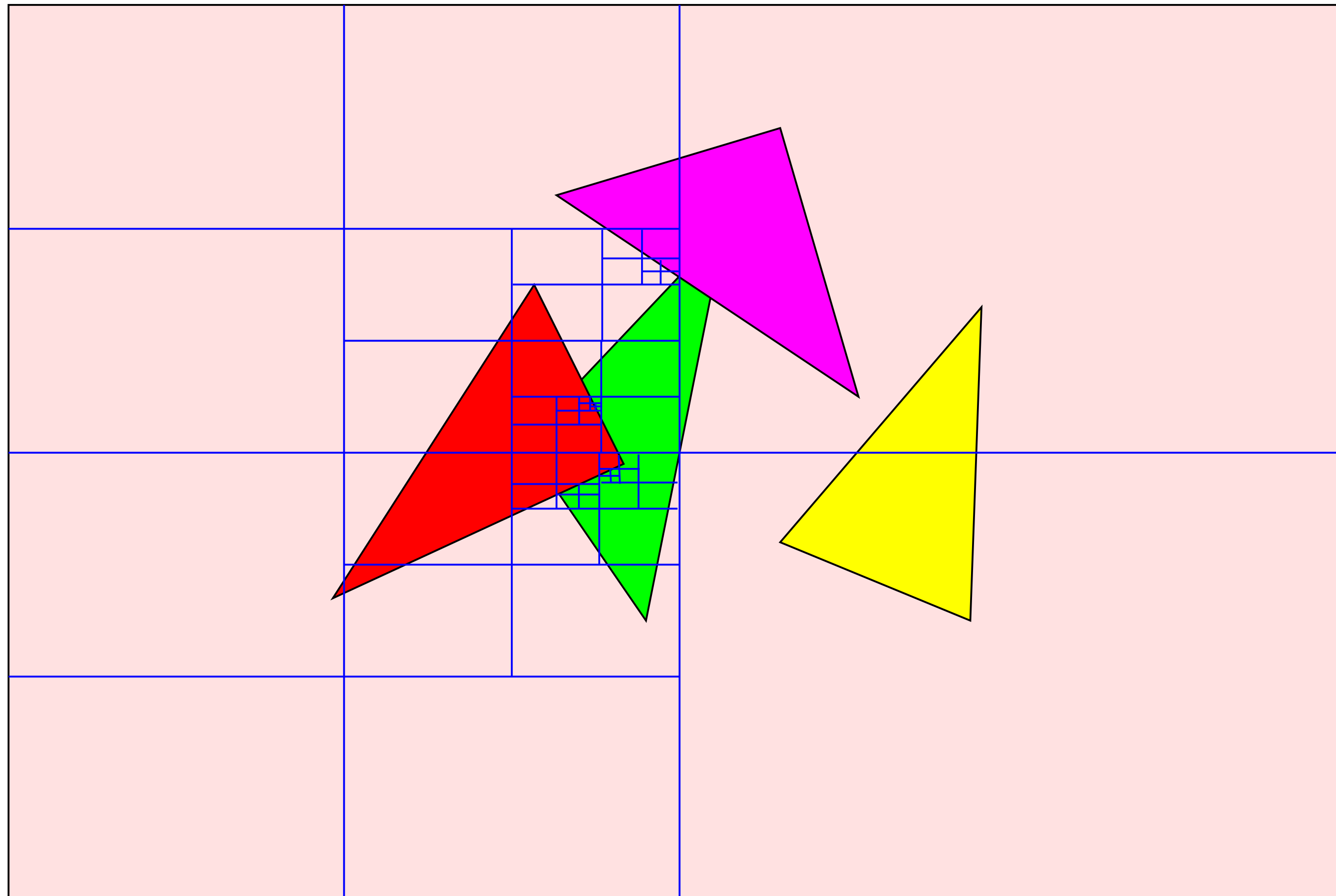
Example



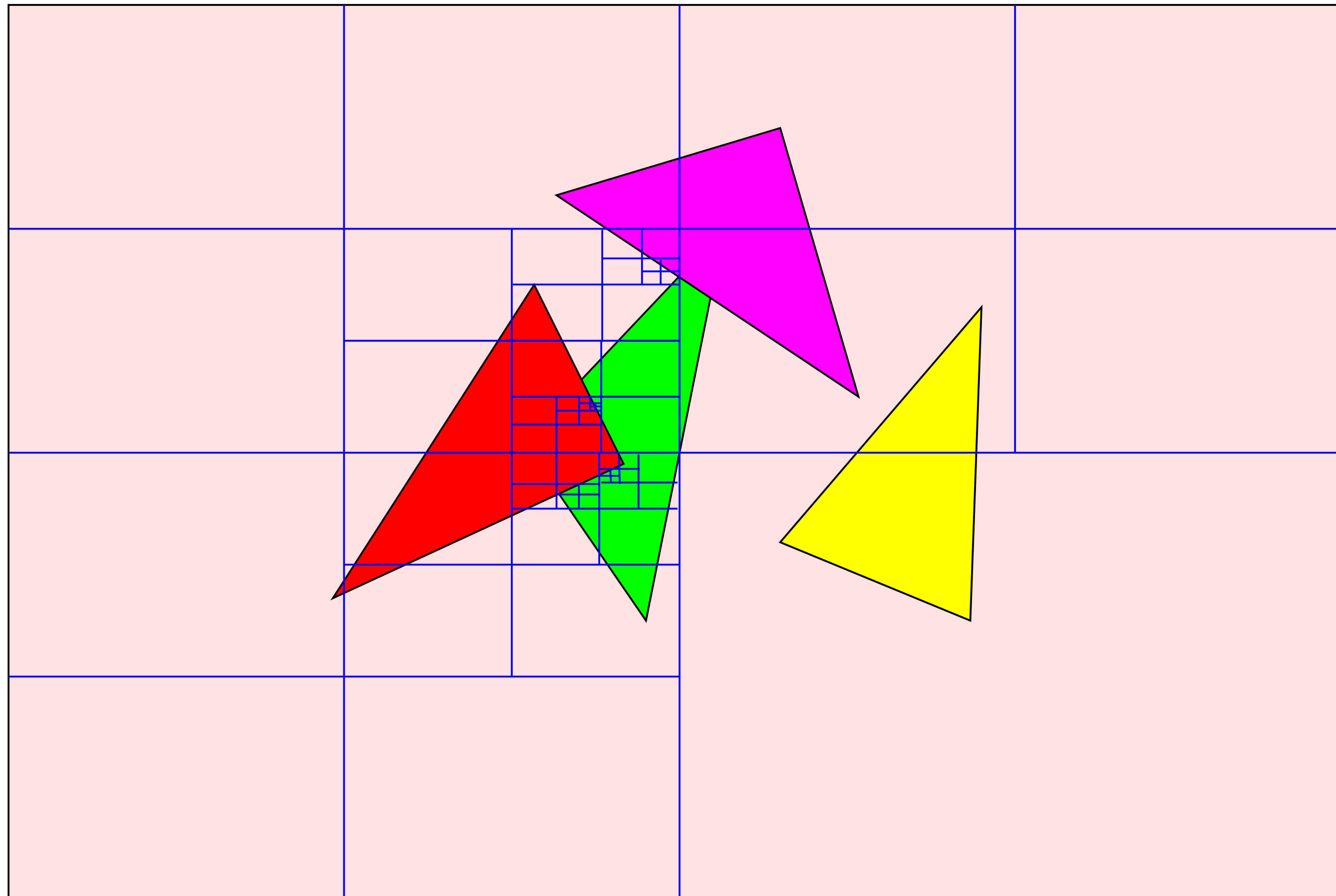
Example



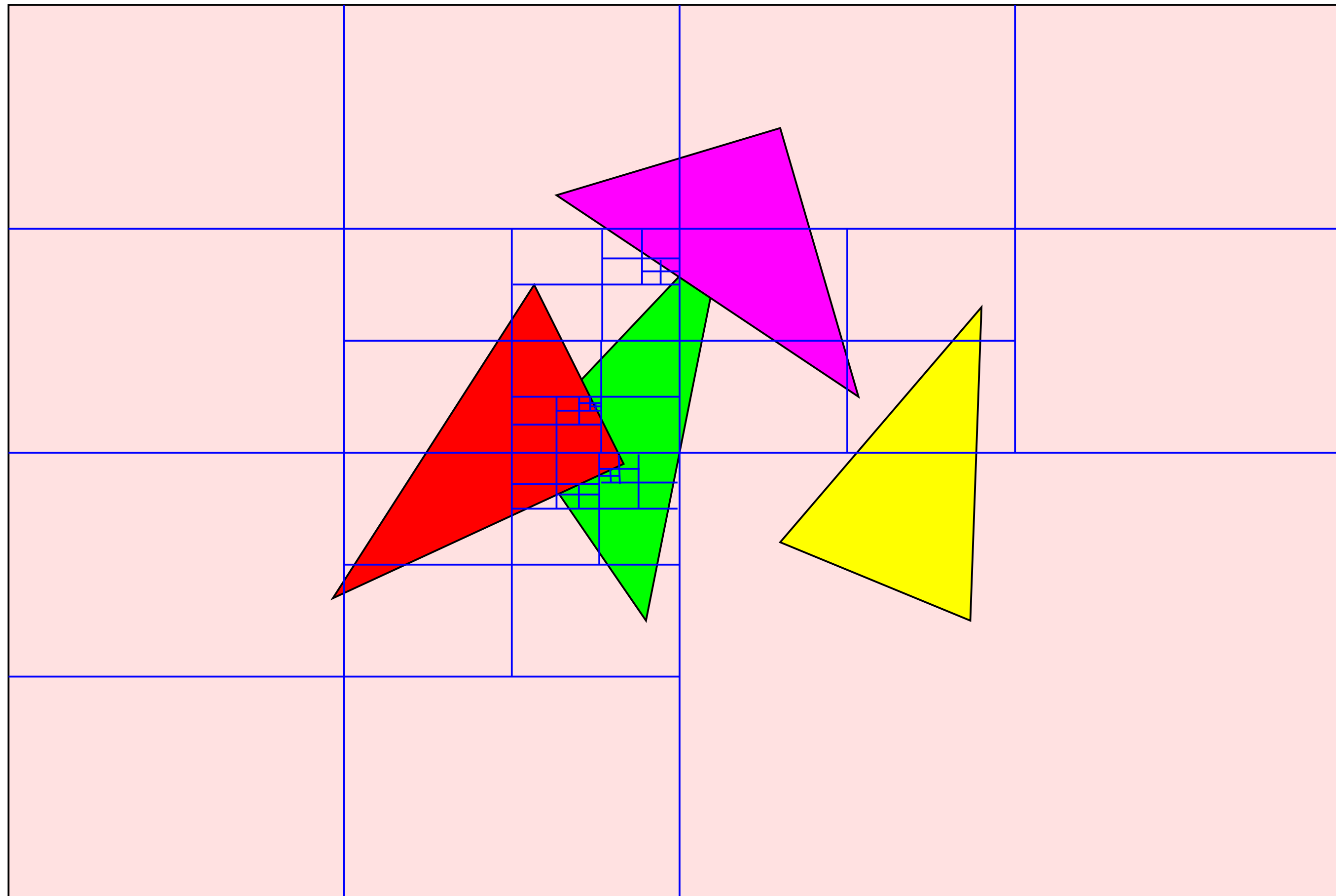
Example



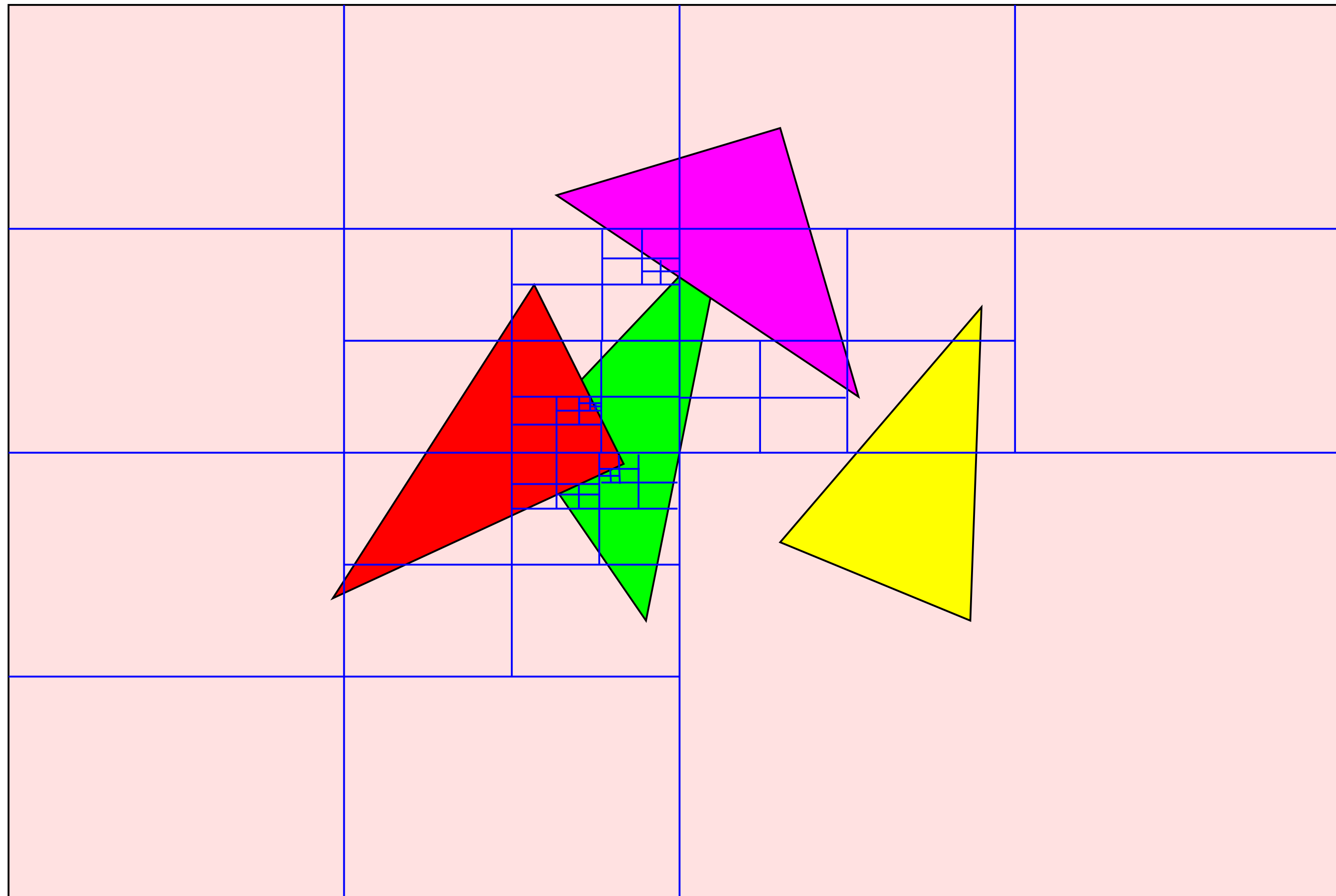
Example



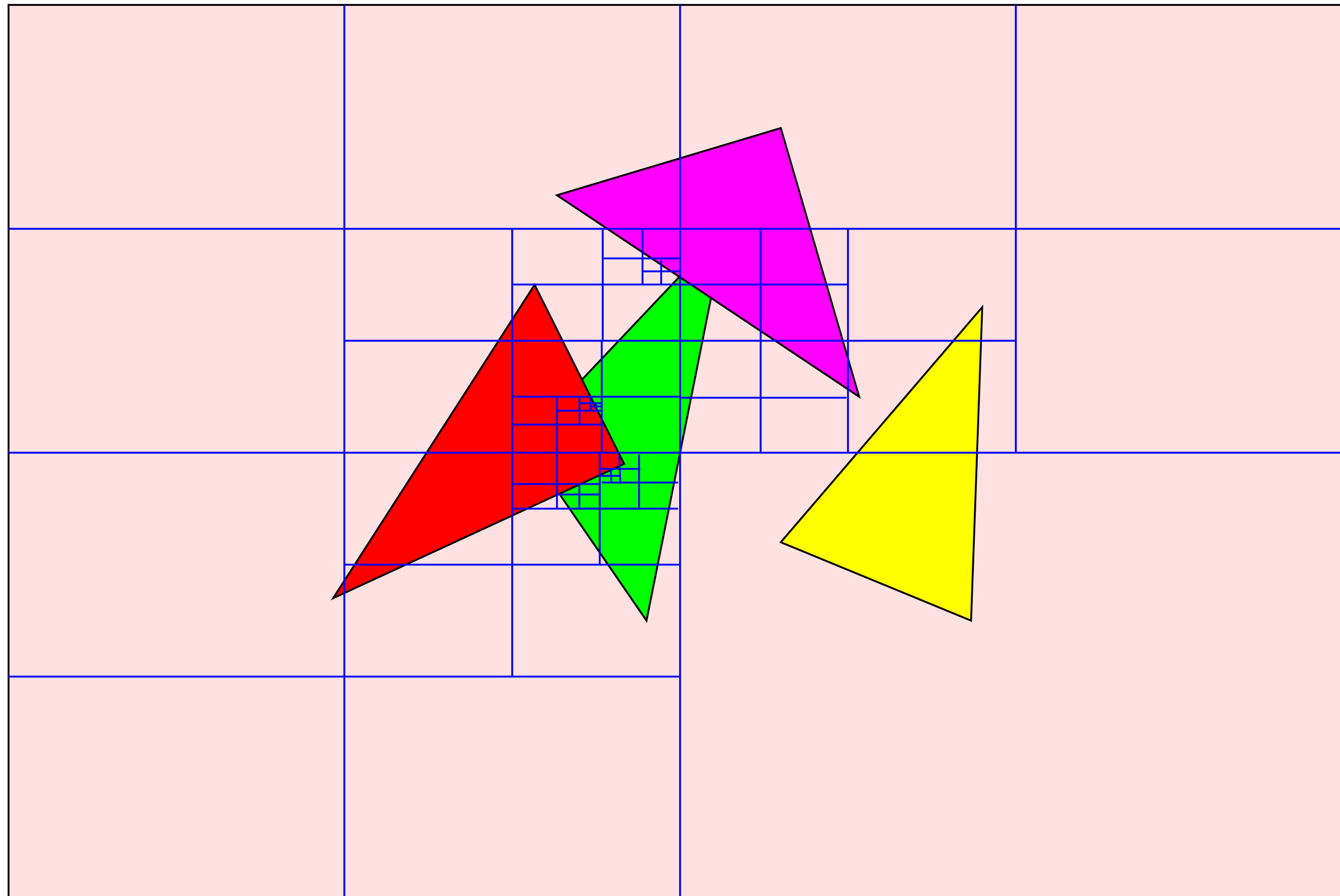
Example



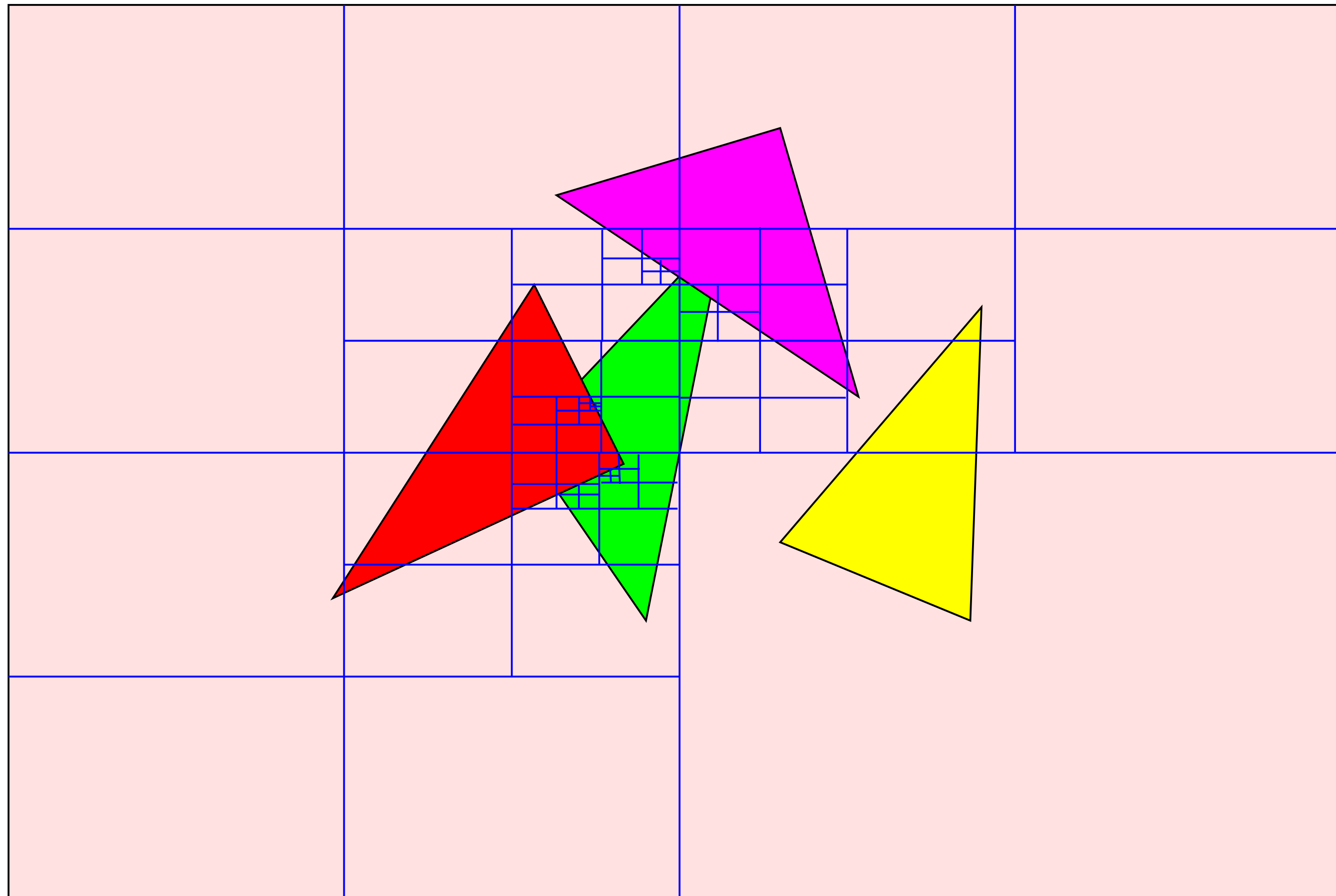
Example



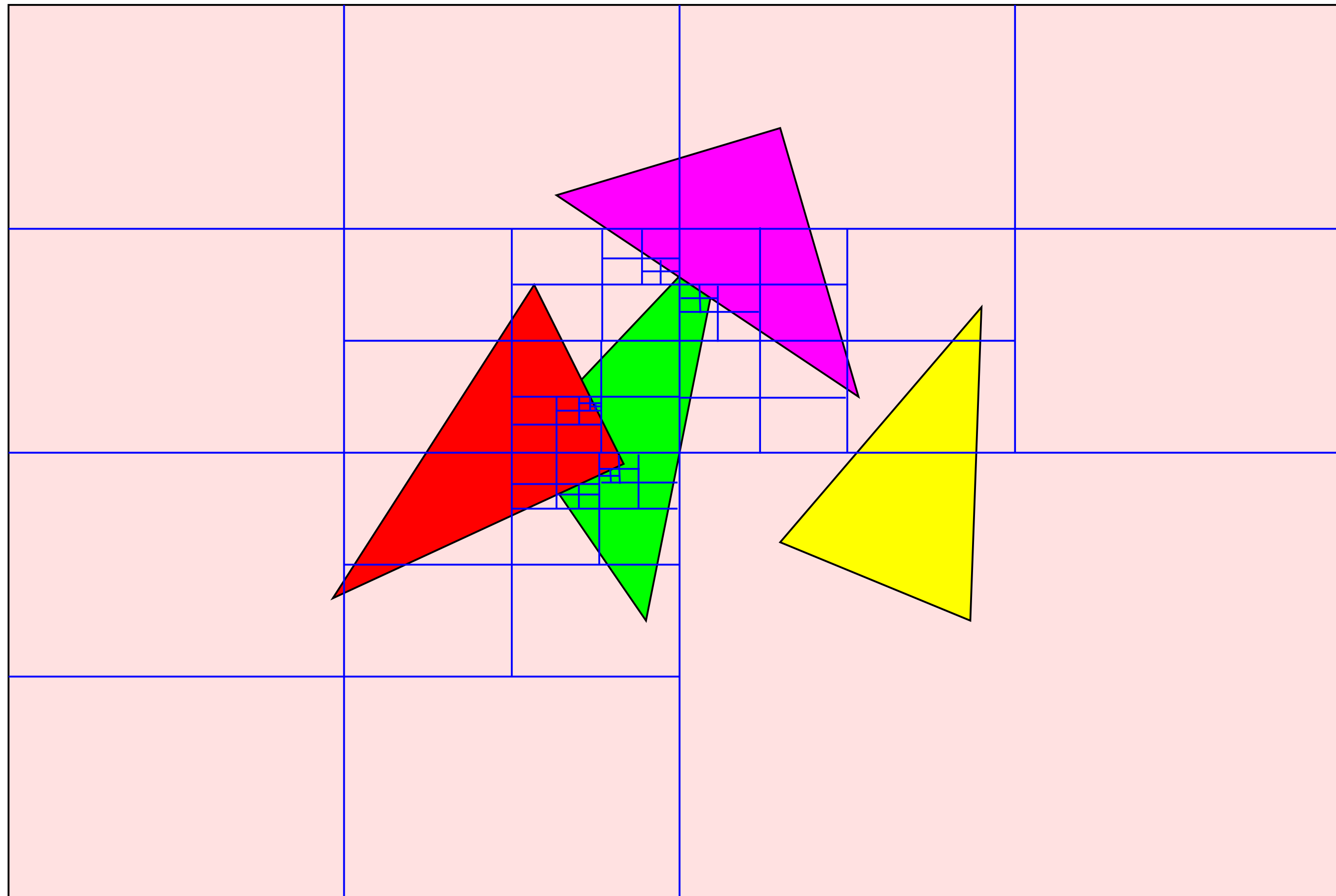
Example



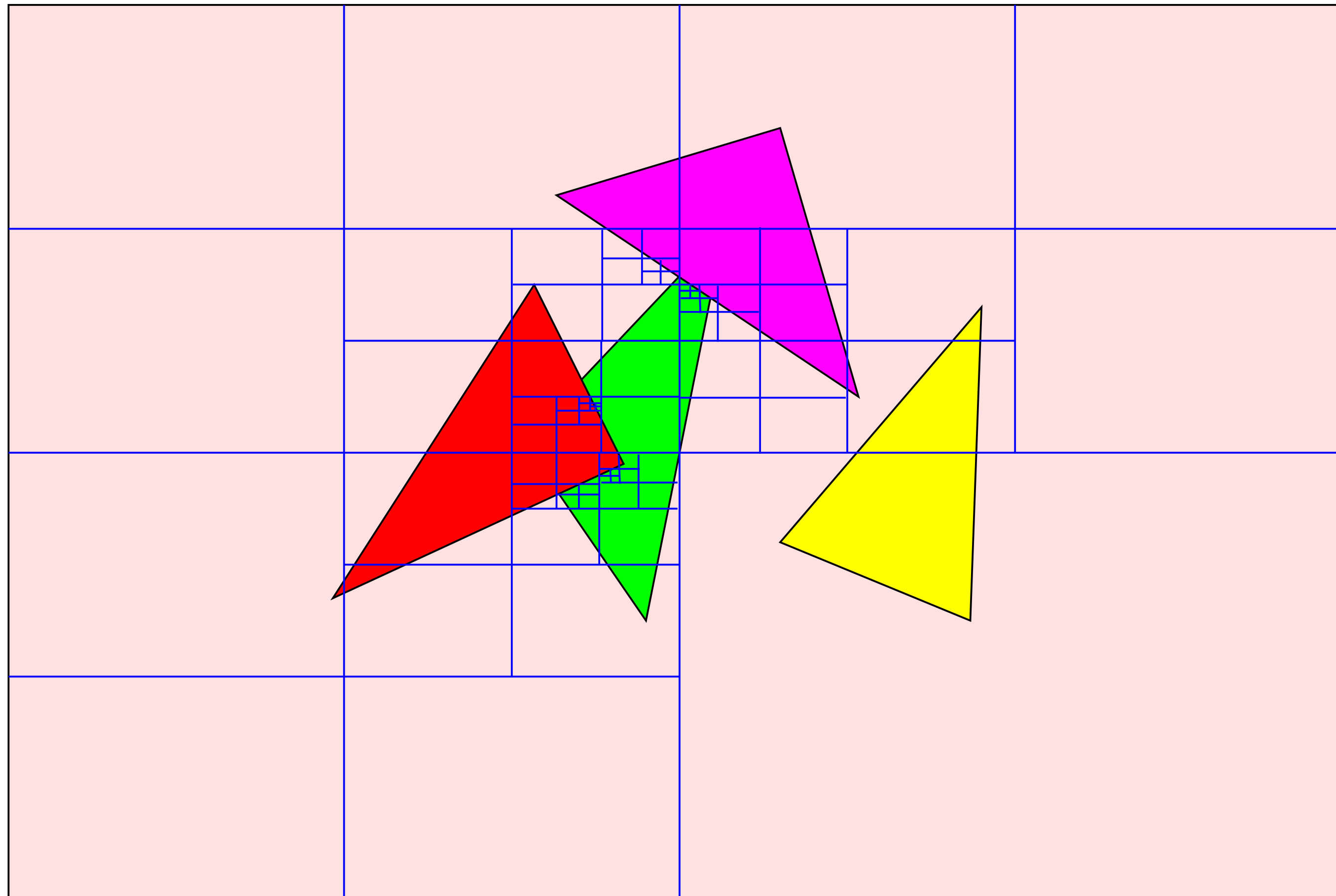
Example



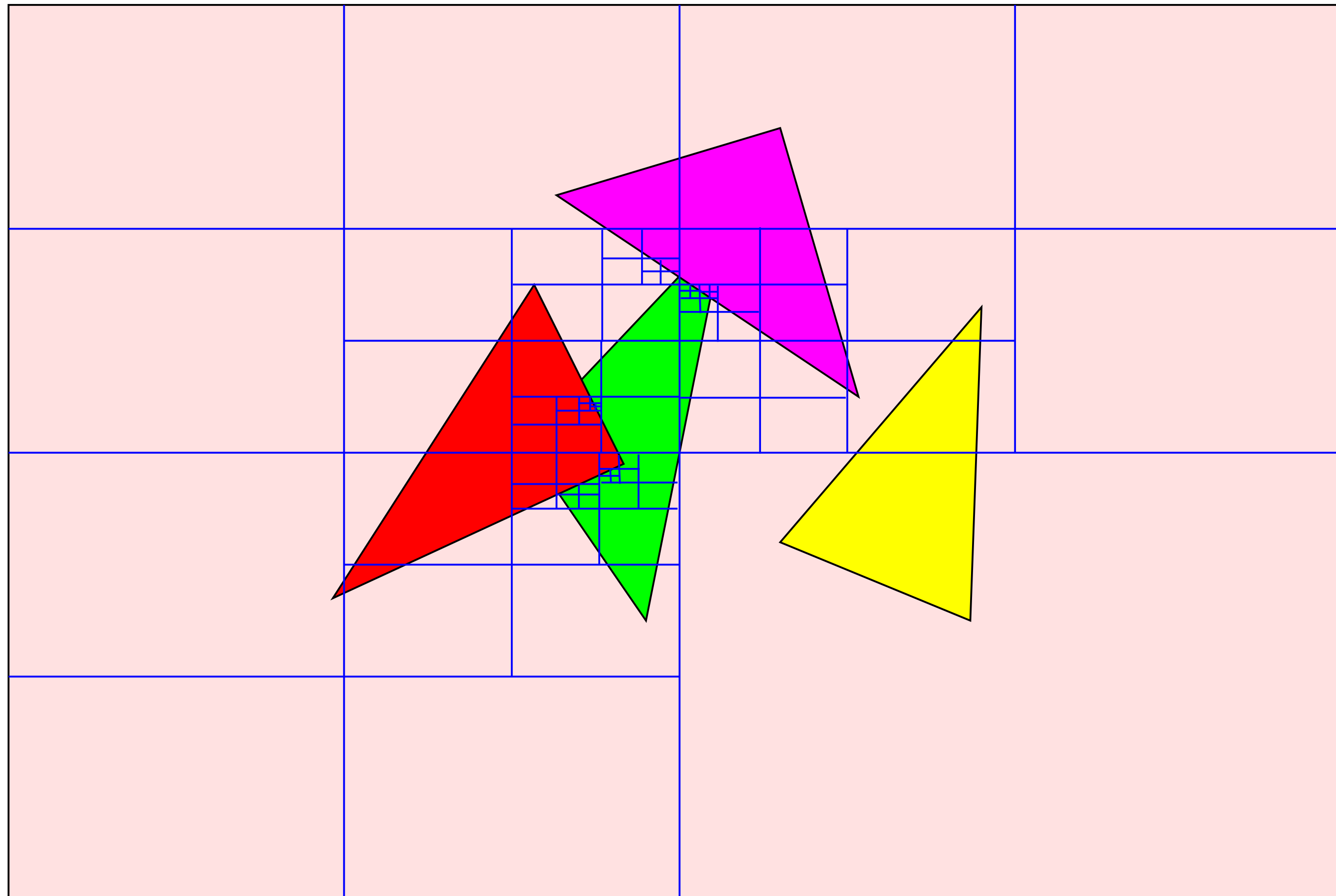
Example



Example



Example



Hidden Line Removal

Vector scan devices

- Draw only the portions of the triangle that should be visible
- Harder than hidden surface removal on raster devices