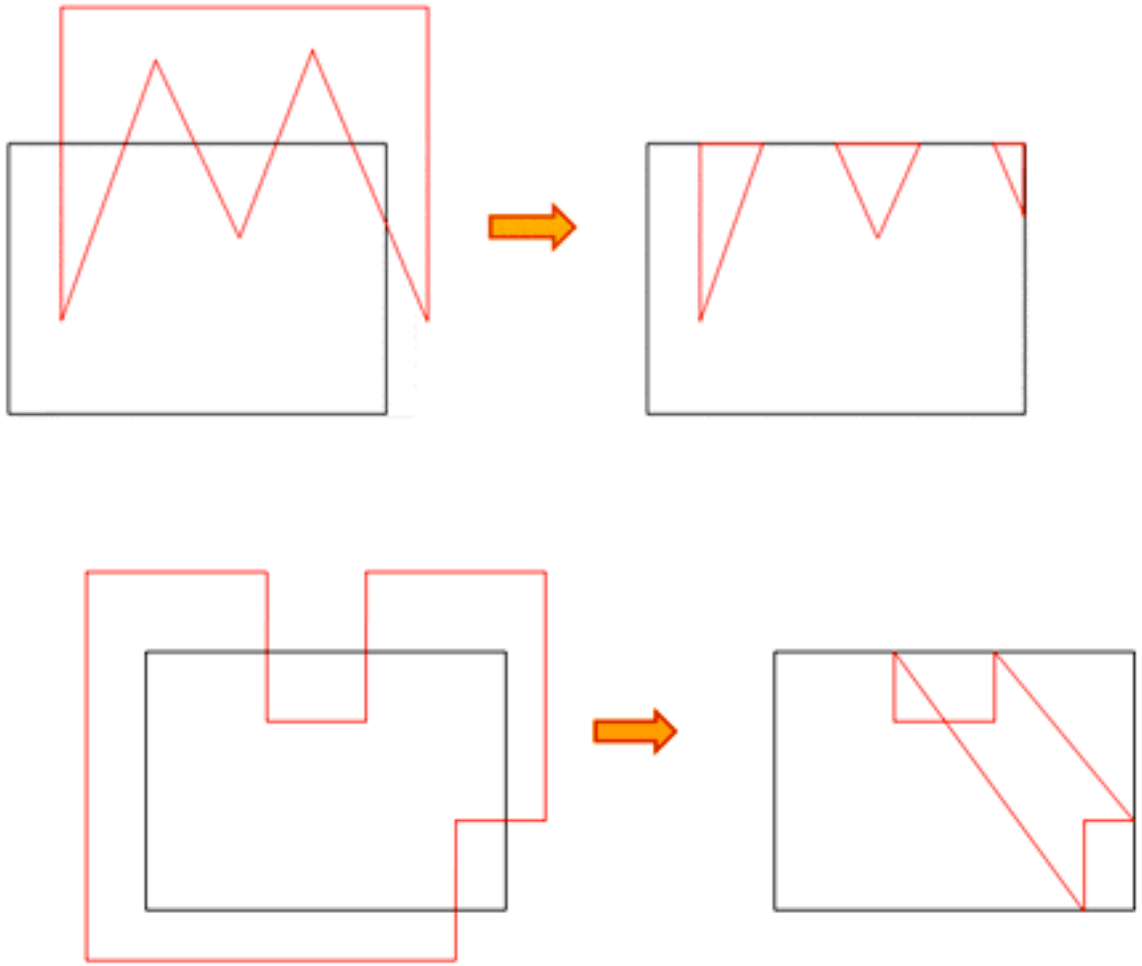
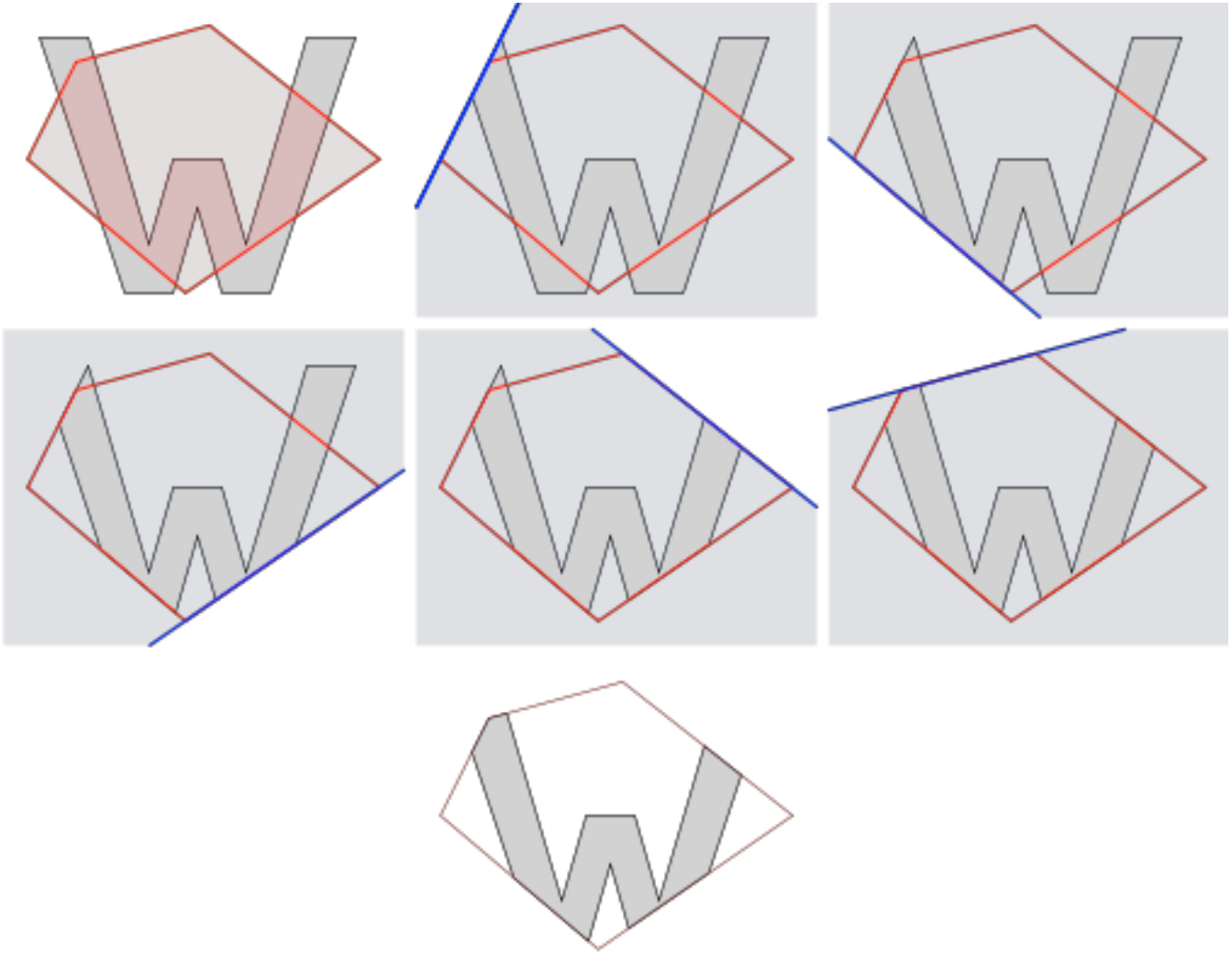
Recorte de objetos: Sutherland-Hodgman



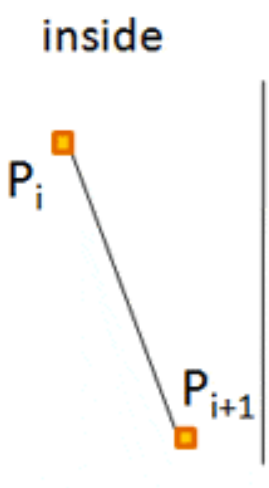
Recorte de objetos: Sutherland-Hodgman



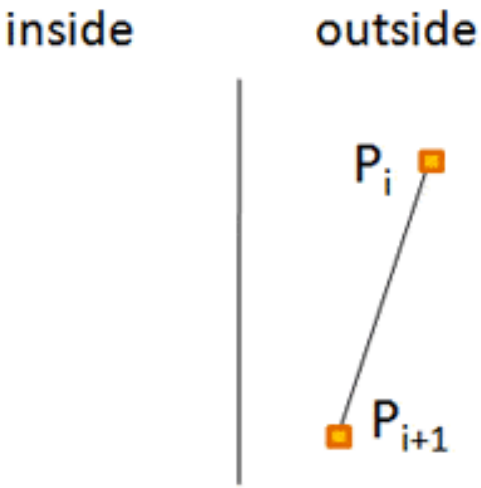
Recorte de objetos: Sutherland-Hodgman

```
List outputList = subjectPolygon;
for (Edge clipEdge in clipPolygon) do
   List inputList = outputList;
   outputList.clear();
   Point S = inputList.last;
   for (Point E in inputList) do
      if (E inside clipEdge) then
         if (S not inside clipEdge) then
            outputList.add(ComputeIntersection(S,E,clipEdge));
         end if
         outputList.add(E);
      else if (S inside clipEdge) then
         outputList.add(ComputeIntersection(S,E,clipEdge));
      end if
      S = E;
   done
done
```

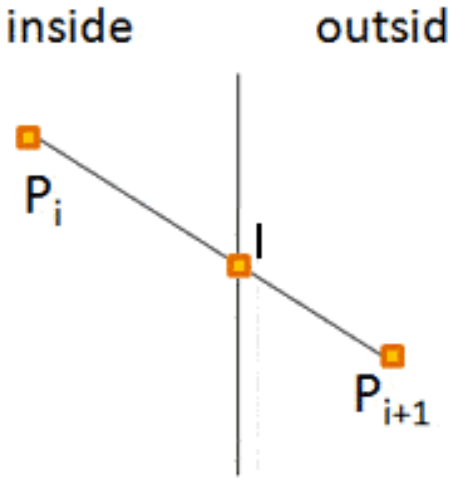
Recorte de objetos: Sutherland-Hodgman



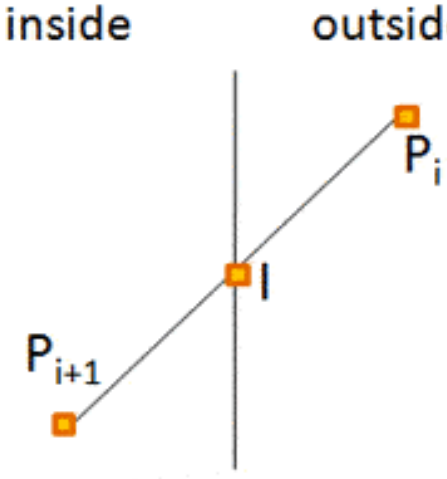
Both points are inside the clipping region
=> add the second one to the result polygon, here it's P_{i+1}.



Both points are outside the clipping region
=> insert none of them to the result polygon.

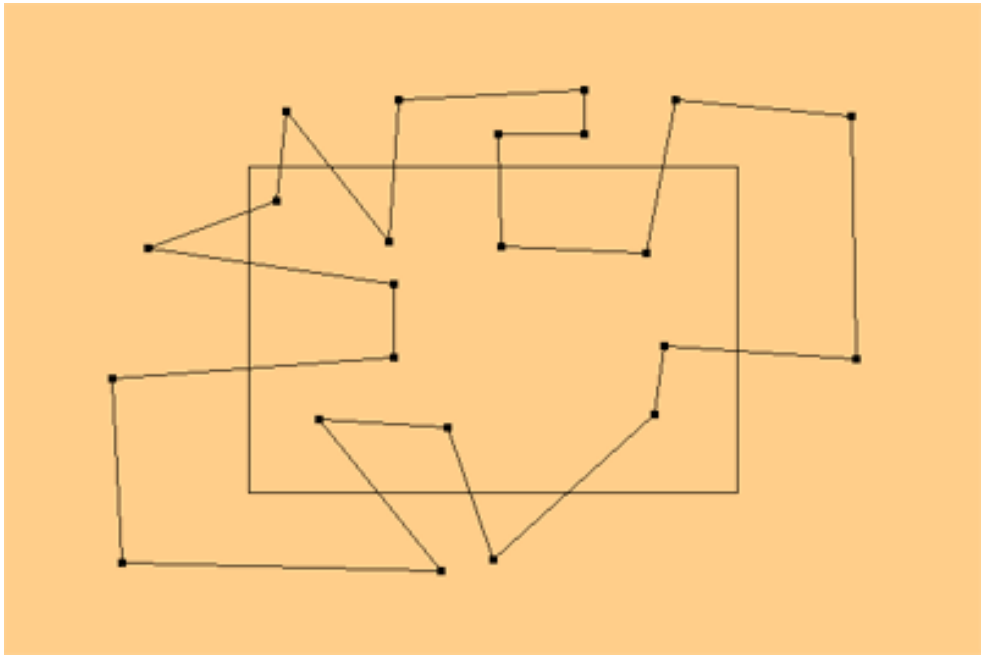


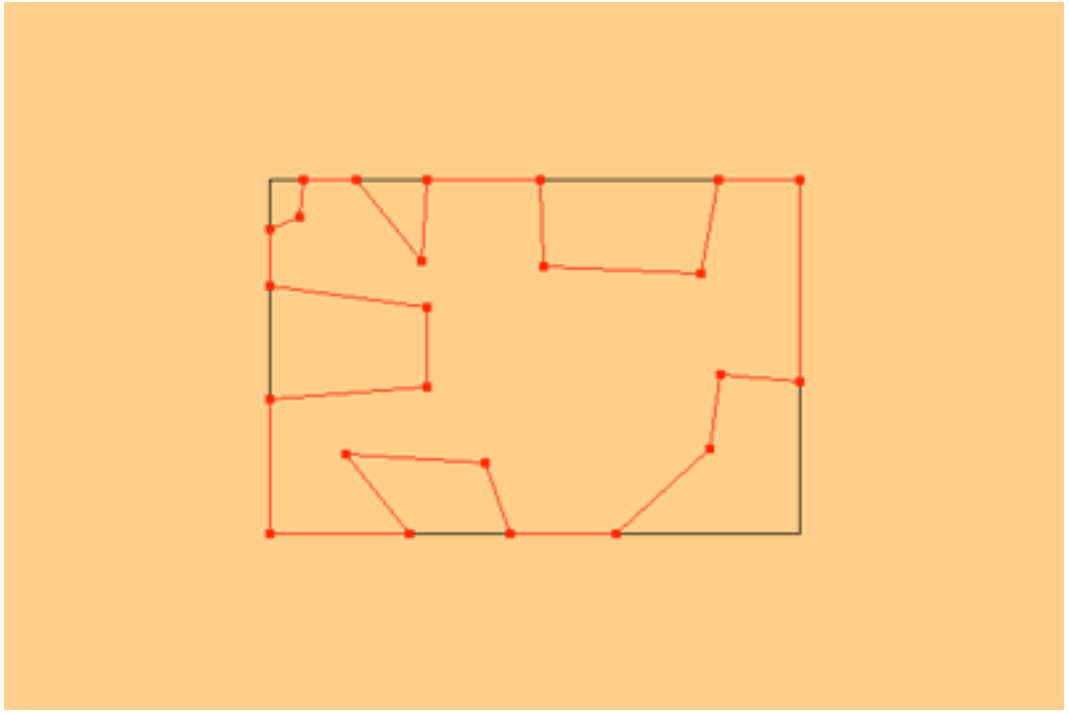
The current point is inside the clipping region but the next one lies outside
=> we add the intersection point I to the result polygon (note that P_i was already inserted in the step before).



The current point is outside the clipping region but the next one lies inside
=> we add the intersection point I and the next point P_{i+1} to the result polygon.

Recorte de objetos: Sutherland-Hodgman





Scan-Line Algorithm

For each scan line:

1. Find the intersections of the scan line with all edges of the polygon.
2. Sort the intersections by increasing x-coordinate.
3. Fill in all pixels between pairs of intersections.

Problem:

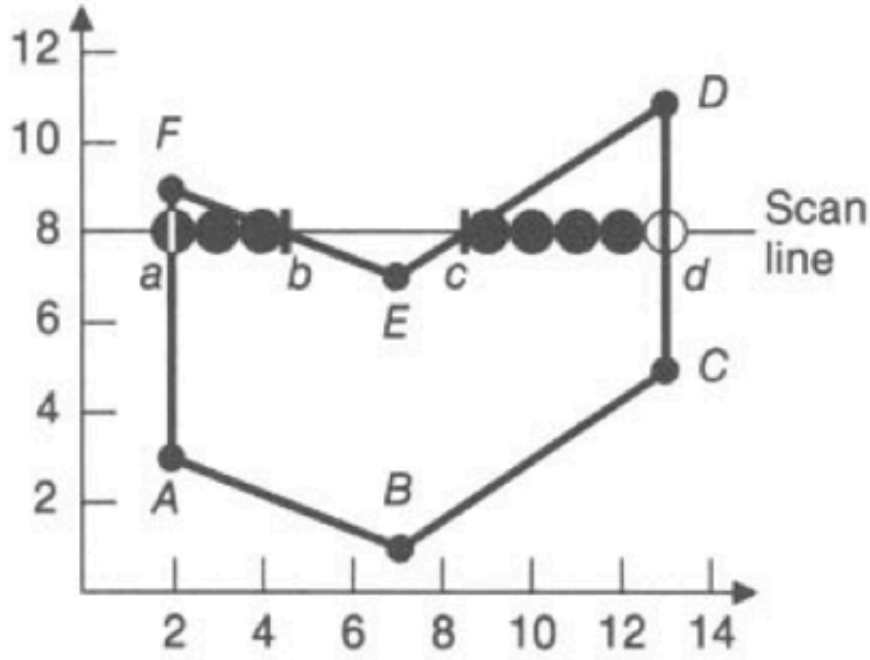
Calculating intersections is slow.

Solution:

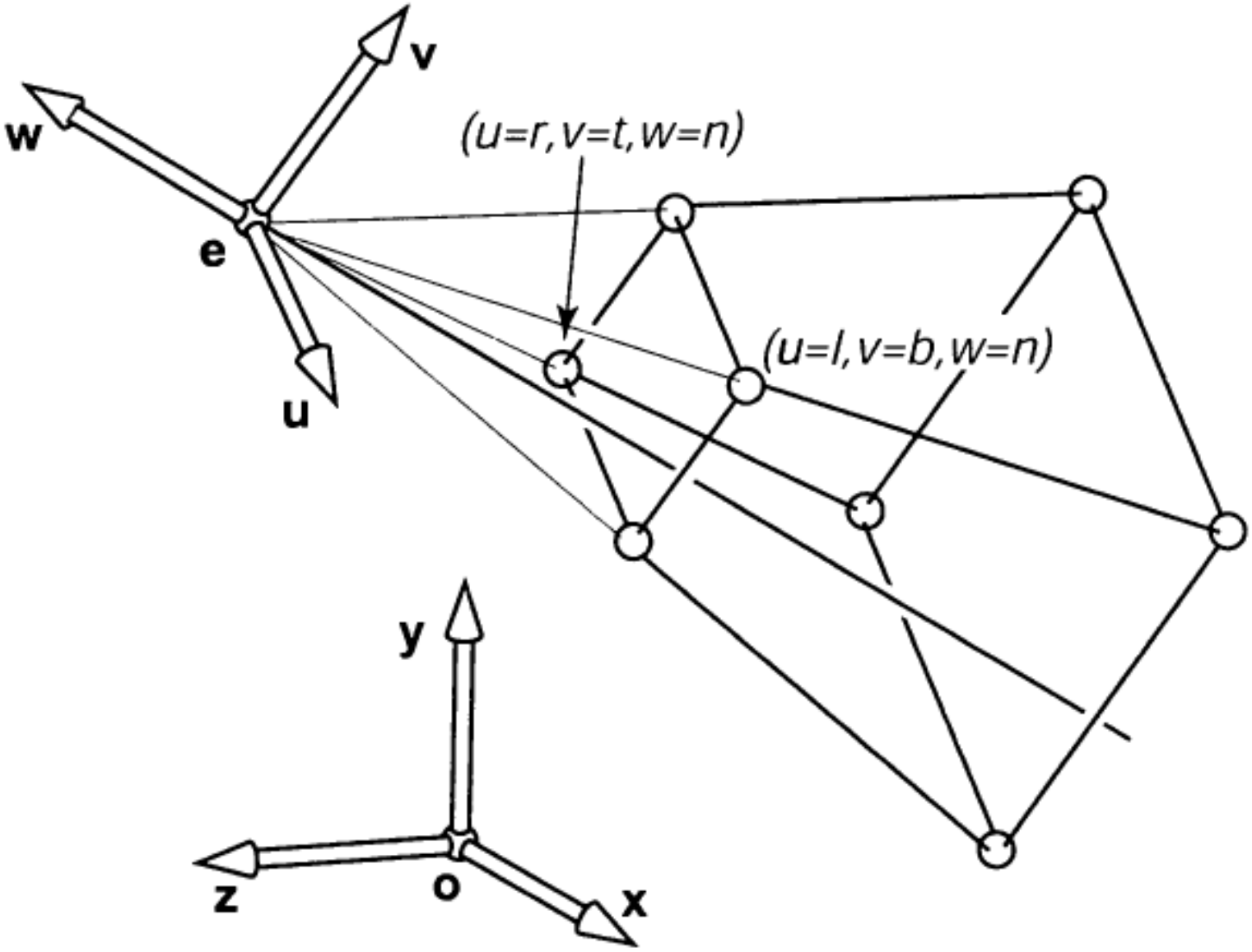
Incremental computation / coherence

For scan line number 8 the sorted list of x-coordinates is (2,4,9,13) (b and c are initially no integers)

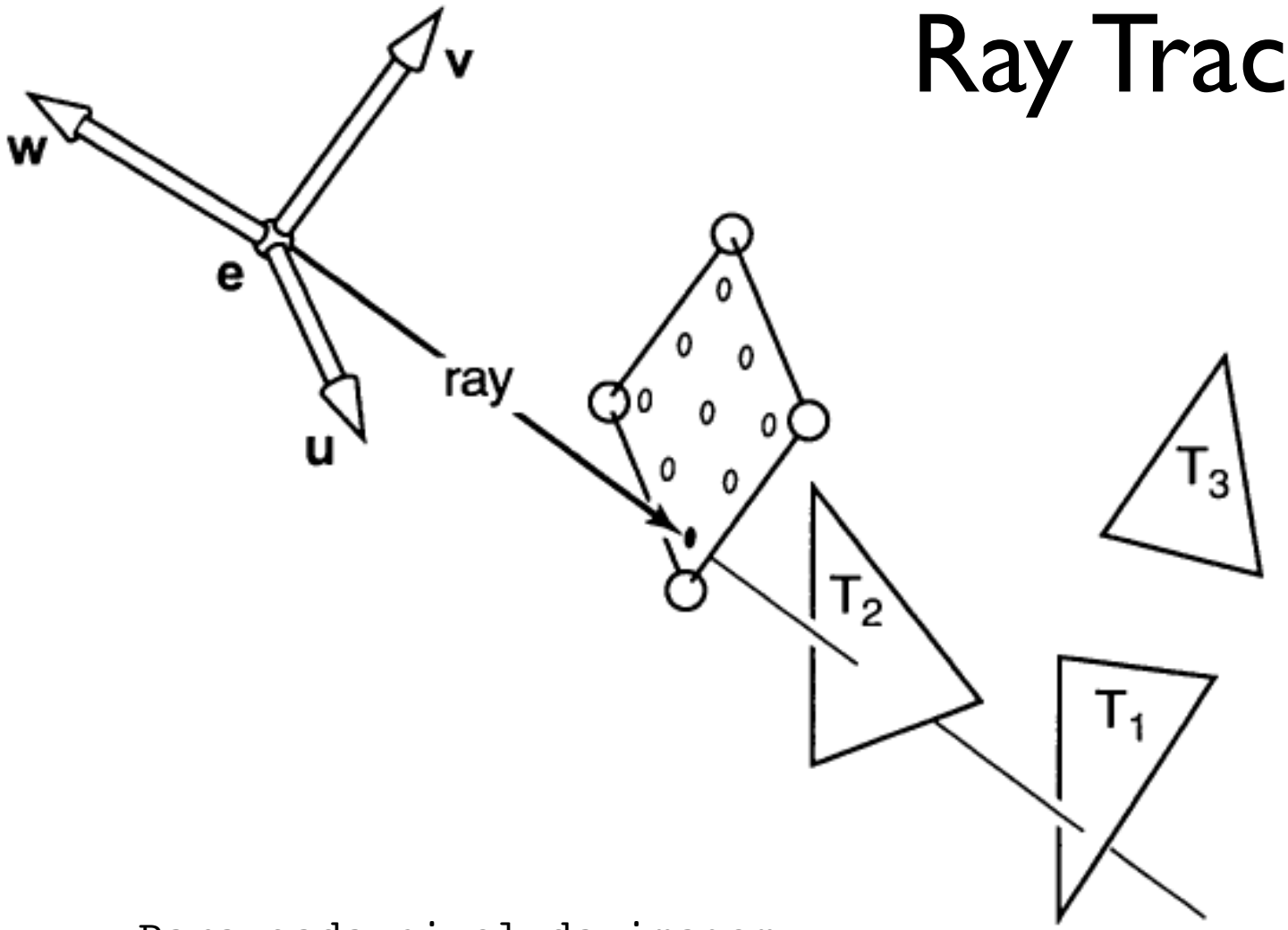
Therefore fill pixels with x-coordinates 2-4 and 9-13.



Ray Tracing

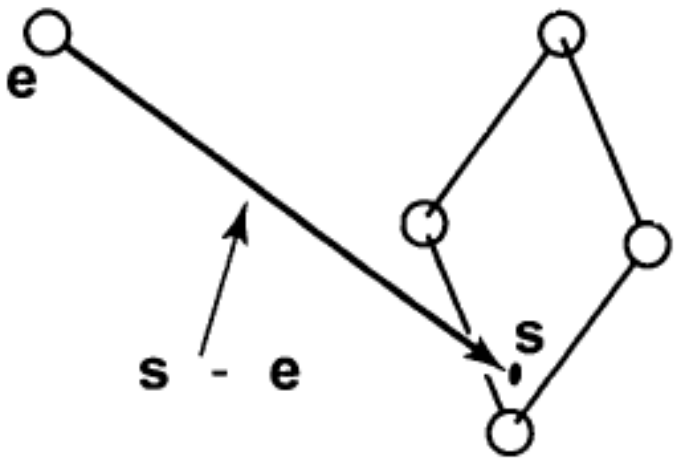


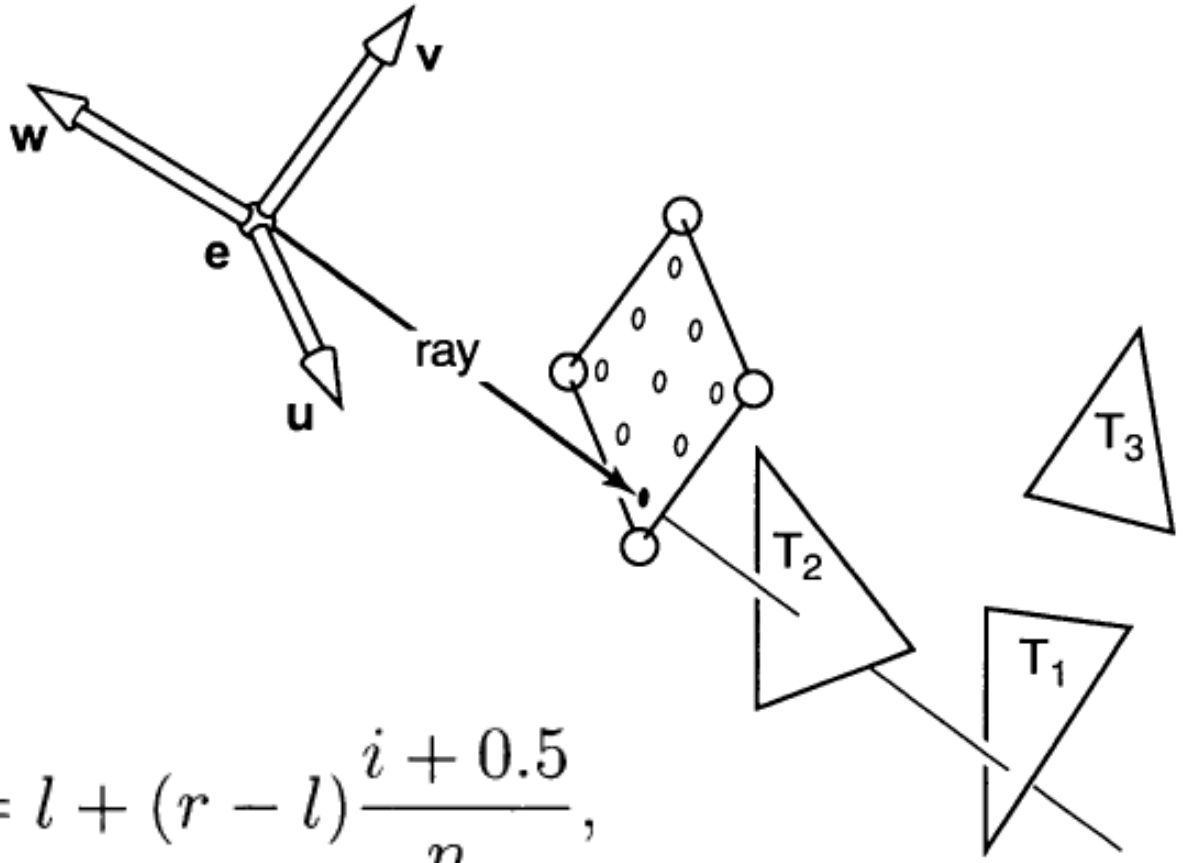
Ray Tracing




```
Para cada pixel da imagem
    Calcule o raio de visão
    Encontre a primeira a intersecção do raio com uma das
superfícies
    Calcule a cor do pixel baseado em material, luz e
normal da superfície
```

Cálculo do raio





$$\mathbf{p}(t) = \mathbf{e} + t(\mathbf{s} - \mathbf{e})$$

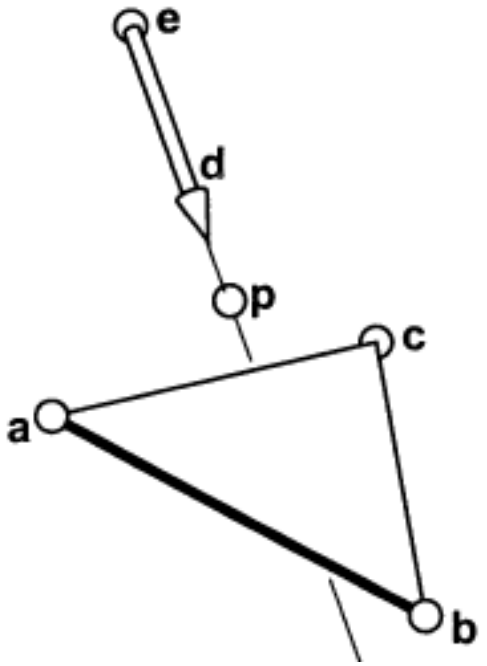
$$u_s = l + (r - l)\frac{i + 0.5}{n_x},$$

$$v_s = b + (t - b)\frac{j + 0.5}{n_y},$$

$$w_s = n$$

$$\mathbf{s} = \mathbf{e} + u_s\mathbf{u} + v_s\mathbf{v} + w_s\mathbf{w}$$

Intersecção raio-triângulo



$$\mathbf{e} + t\mathbf{d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$

$$\beta > 0, \gamma > 0$$

$$\beta + \gamma < 1$$

$$\begin{aligned} x_e + tx_d &= x_a + \beta(x_b - x_a) + \gamma(x_c - x_a), \\ y_e + ty_d &= y_a + \beta(y_b - y_a) + \gamma(y_c - y_a), \\ z_e + tz_d &= z_a + \beta(z_b - z_a) + \gamma(z_c - z_a). \end{aligned}$$

Algoritmo recursivo

```
{
  Criar um raio a partir do ponto de visão (observador) e que passe pelo PixelActual
  Inicializar NearestT em INFINITO
  Inicializar NearestObject em NULL
  Para cada objecto da cena
  {
     Se o raio intercepta o ObjectoActual
     {
        Se a distância t da intercepção é menor que NearestT
        {
           Colocar NearestT = t distância da intercepção
           Colocar NearestObject = ObjectoActual
        }
     }
  }
  Se NearestObject = NULL
  {
     Preencher PixelActual com a cor do fundo
  }
  Senão
  {
     Lançar um novo raio a cada fonte de luz para comprovar as sombras
     Se a superfície é reflectora
     {
        gerar um raio reflector (recursivo)
     }
     Se a superfície é transparente
     {
        gerar um raio refractor (recursivo)
     }
     Usar NearestObject e NearestT para processar a função sombreado
     Preencher este pixel com a cor resultante da função sombreado
  }
}
```