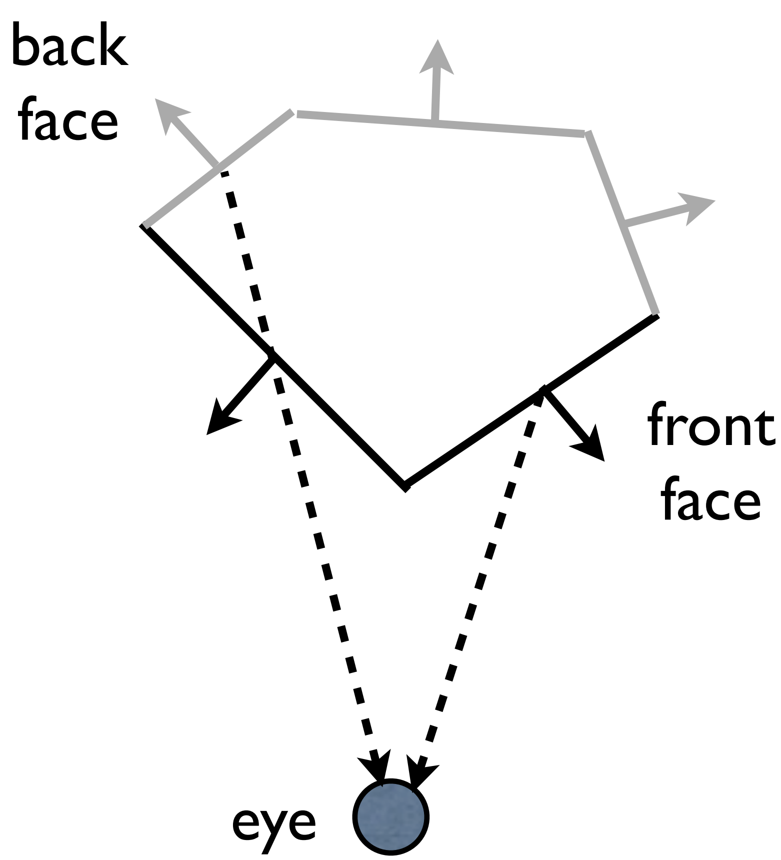
Computer Graphics

CSC 418/2504

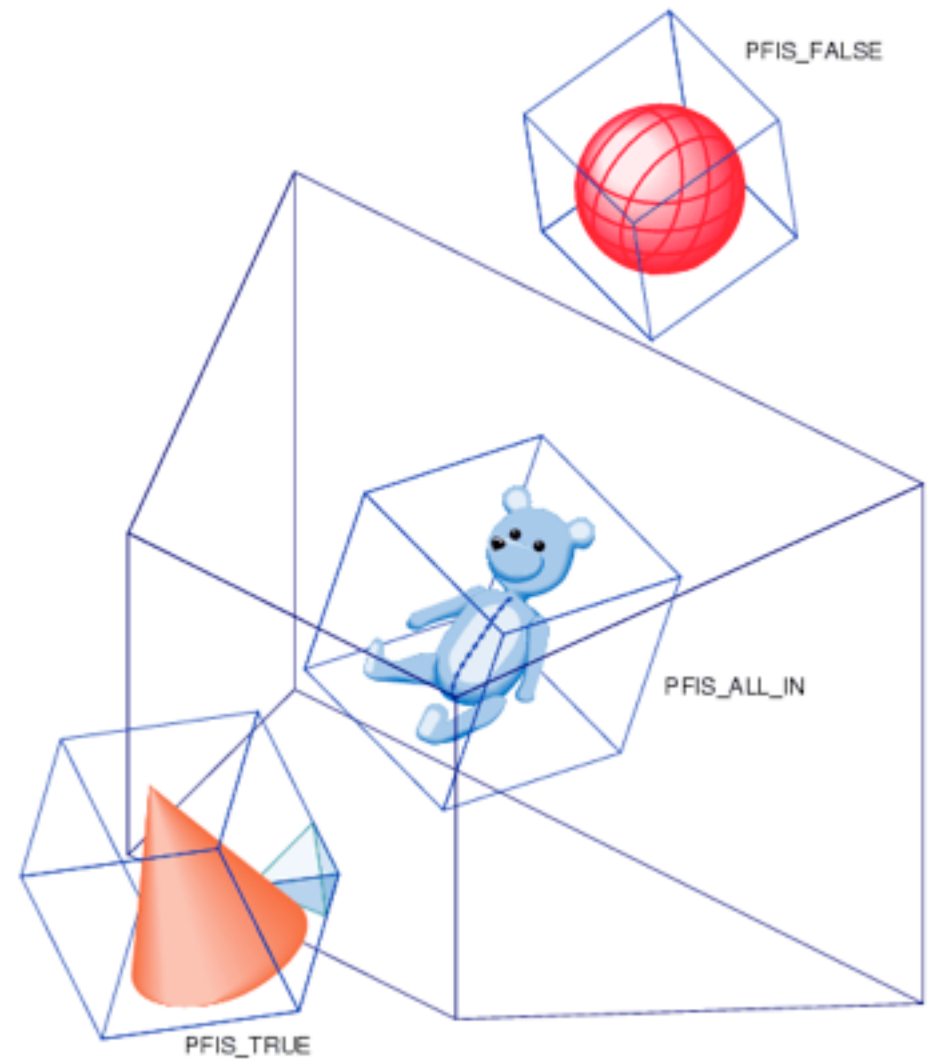
Patricio Simari
October 26, 2011

Some slides courtesy of Patrick Coleman
Some figures courtesy of Peter Shirley,
"Fundamentals of Computer Graphics", 2nd Ed.

Culling



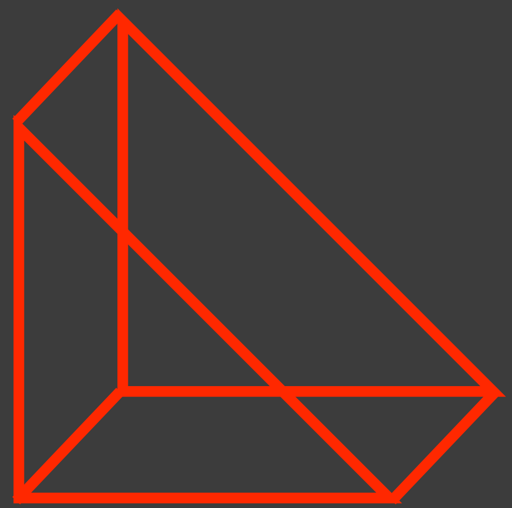
Back faces

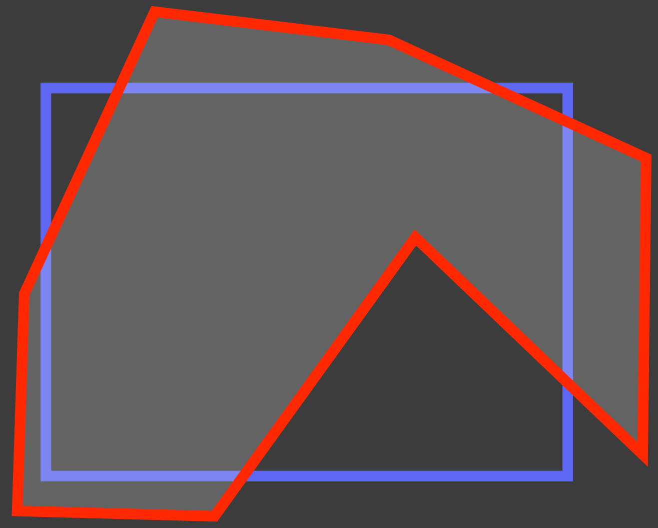


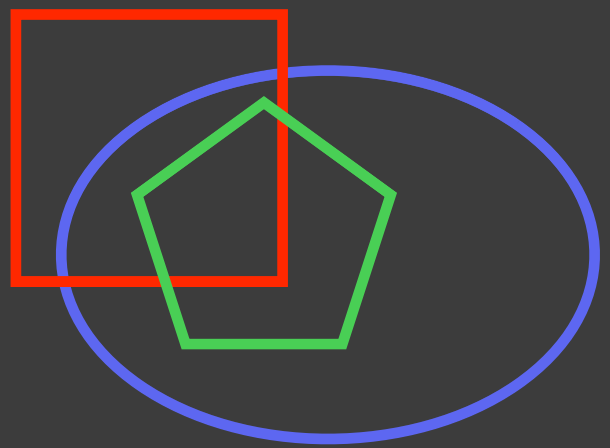
Bounding Volumes

google images

Visibility Review



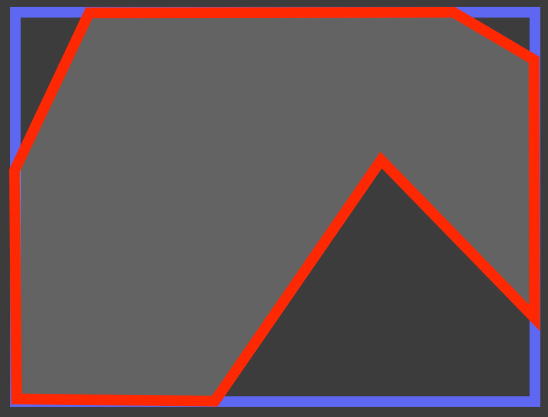


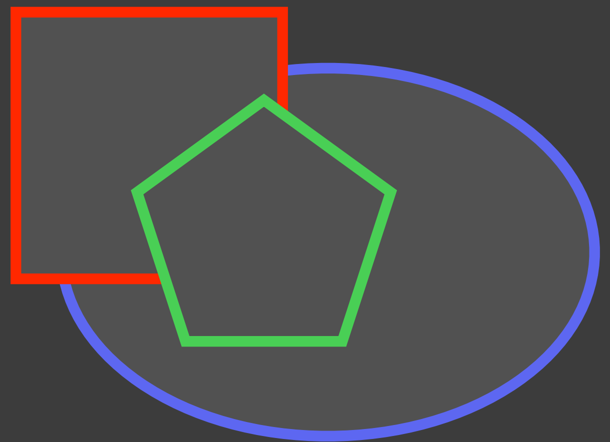


Backface Culling

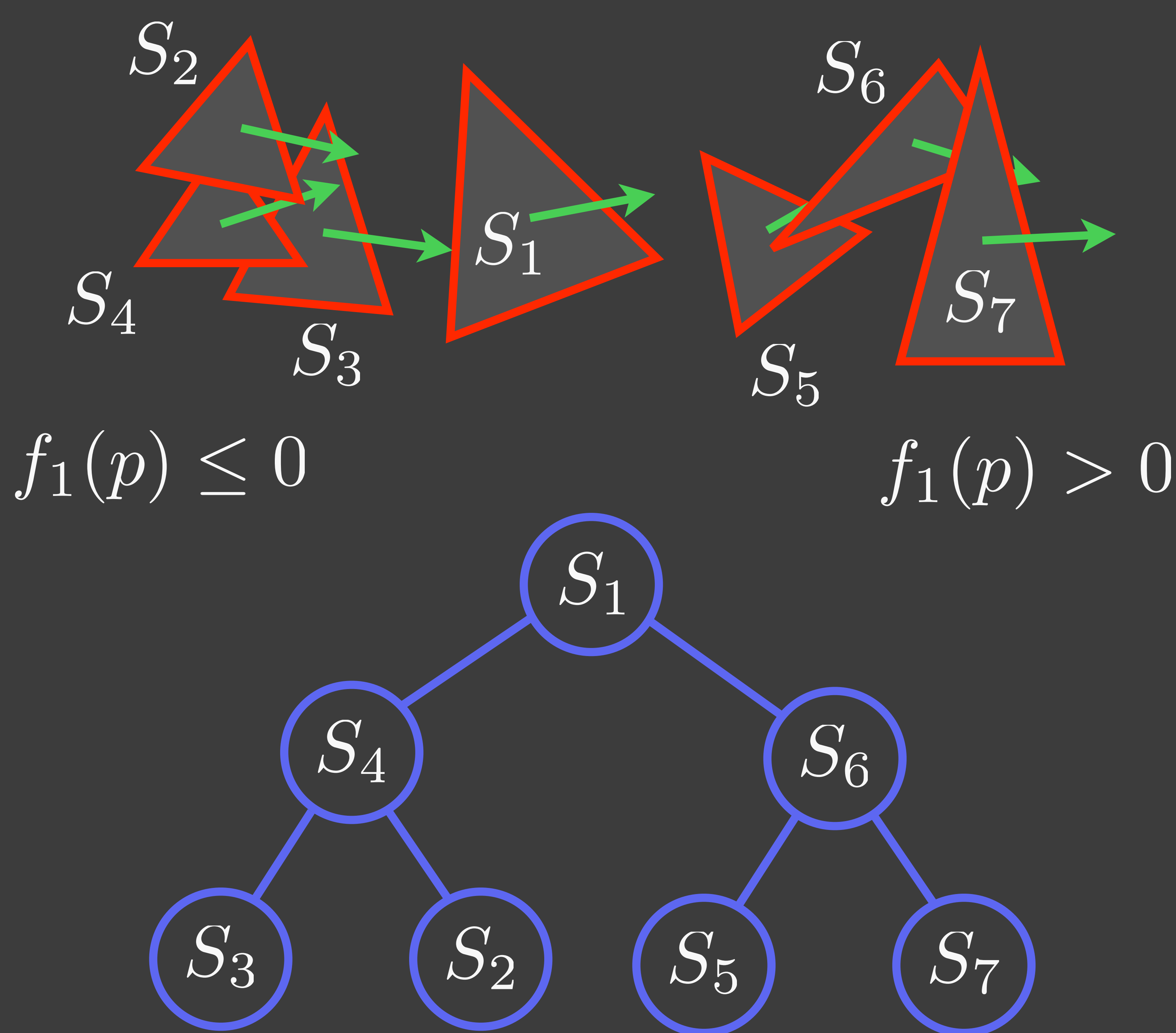
Clipping

Shape Occlusion



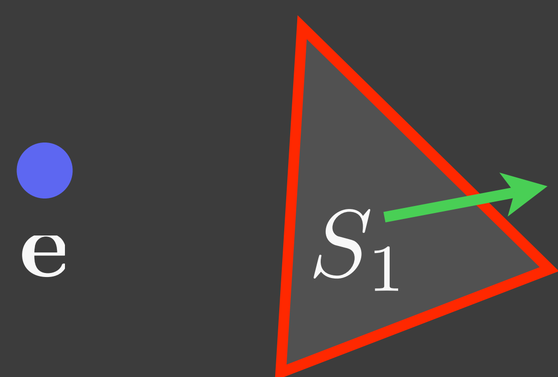


BSP Tree Review



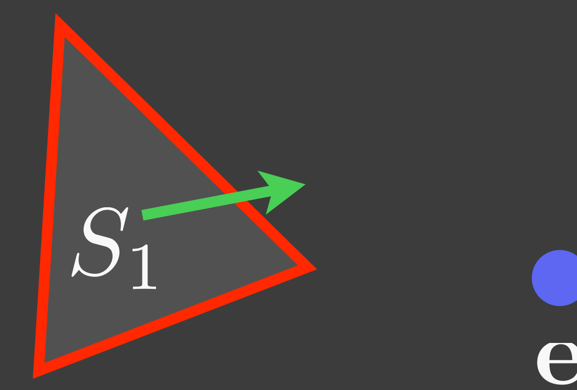
Drawing a BSP Tree

Principle: Draw the opposite side of the eye first.



$f_1(\mathbf{e}) \leq 0$

draw right subtree
draw S_1
draw left subtree



$f_1(\mathbf{e}) > 0$

draw left subtree
draw S_1
draw right subtree

Drawing Pseudocode

```
class Node():
  Shape shape
  Node left
  Node right


class Tree():
  Node root
  draw()
  add()
```

```
Tree.draw(n, e):
  if empty(node):
    return
  if f_n(e) <= 0:
    self.draw(n.right)
    draw n.shape
    self.draw(n.left)
  else:
    self.draw(n.left)
    draw n.shape
    self.draw(n.right)
```

Creating a BSP Tree

Given a shape sequence

$$[S_1, S_2, ..., S_n]$$

```
tree.root = node(S_1)
for i in [2, n]:
  tree.add(S_i, tree.root)
```

```
Tree.add(s, n):
```

Add left

```
if $f_n(\mathbf{p}) \leq 0 \; \forall \mathbf{p} \in s$:
  if n.left is None:
    node.left = Node(s)
  else
    self.add(s, node.left)
```

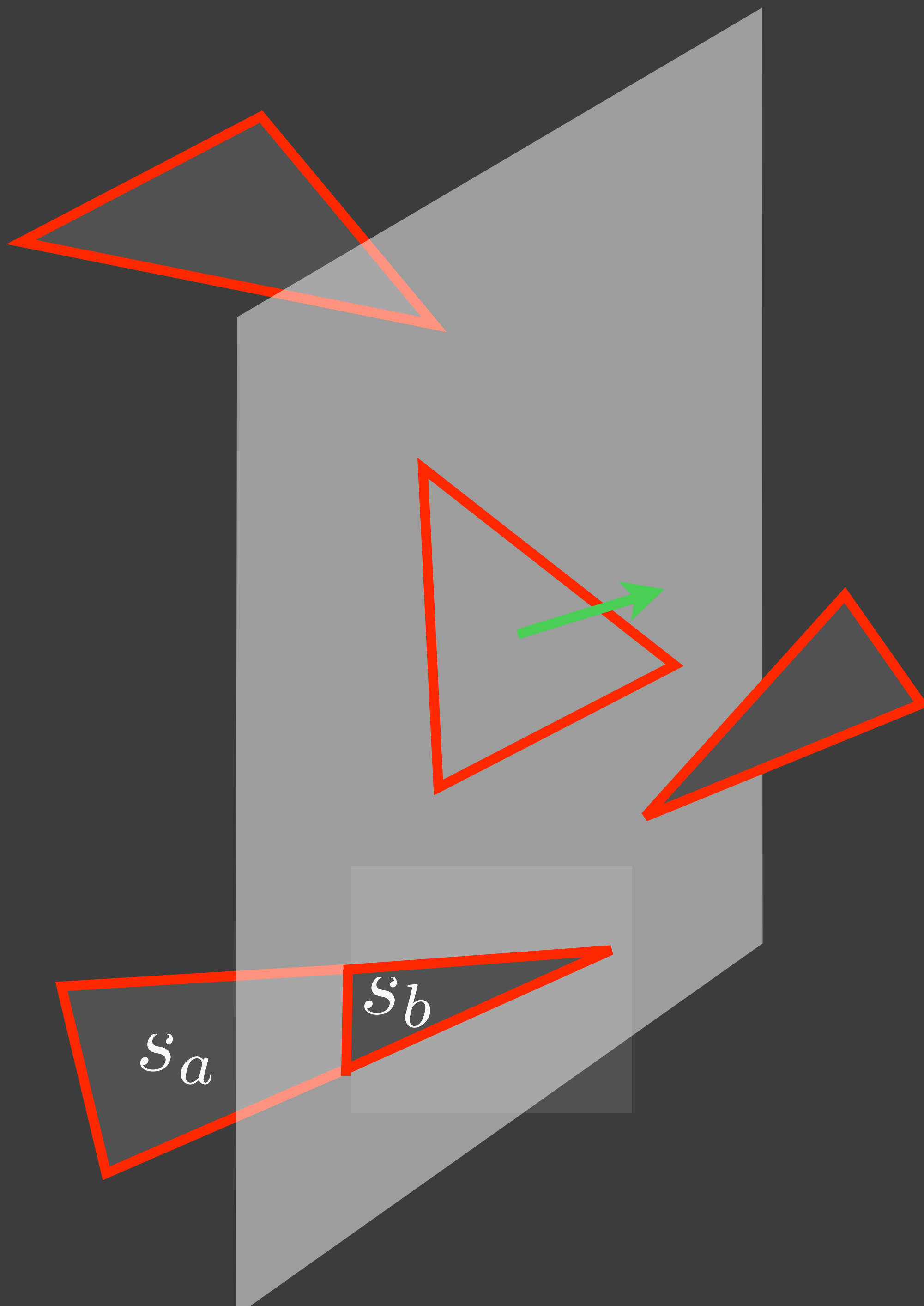
Add right

```
else if $f_n(\mathbf{p}) \geq 0 \; \forall \mathbf{p} \in s$:
  if n.right is None:
    node.right = Node(s)
  else
    self.add(s, node.right)
```

Split s

```
else:
  split s into s_a and s_b s.t.:
      $f_n(\mathbf{p}) \geq 0 \; \forall \mathbf{p} \in s_b$
      $f_n(\mathbf{p}) \leq 0 \; \forall \mathbf{p} \in s_a$
  self.add(s_a, n)
  self.add(s_b, n)
```

Creating a BSP Tree



Tree.add(s, n):

Add left

if $f_n(\mathbf{p}) \leq 0 \ \forall \mathbf{p} \in s$:
  if n.left is None:
    node.left = Node(s)
  else
    self.add(s, node.left)

Add right

else if $f_n(\mathbf{p}) \geq 0 \ \forall \mathbf{p} \in s$:
  if n.right is None:
    node.right = Node(s)
  else
    self.add(s, node.right)

Split s

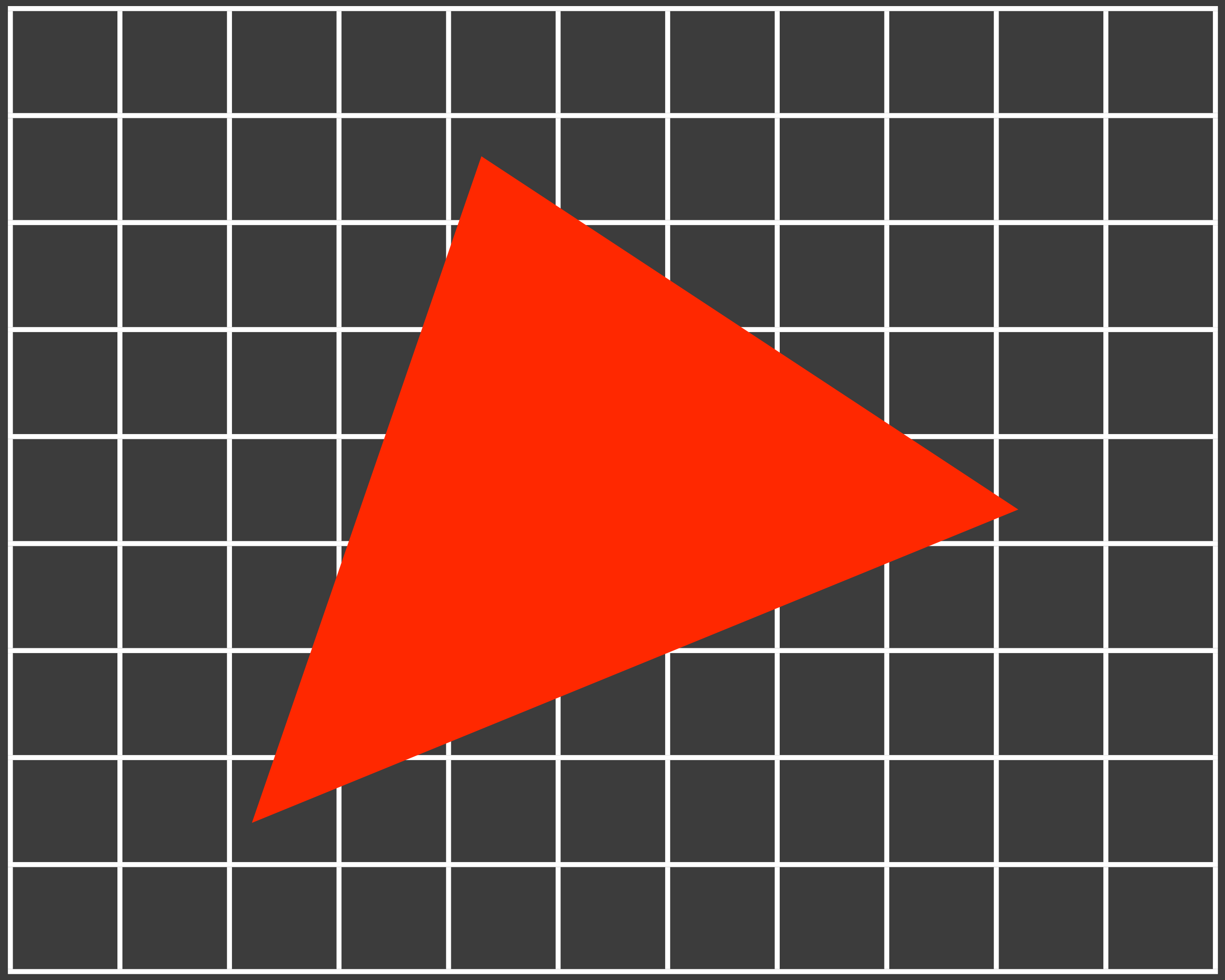
else:
  split s into s_a and s_b s.t.:

$$f_n(\mathbf{p}) \geq 0 \ \forall \mathbf{p} \in s_b$$
$$f_n(\mathbf{p}) \leq 0 \ \forall \mathbf{p} \in s_a$$

  self.add(s_a, n)
  self.add(s_b, n)

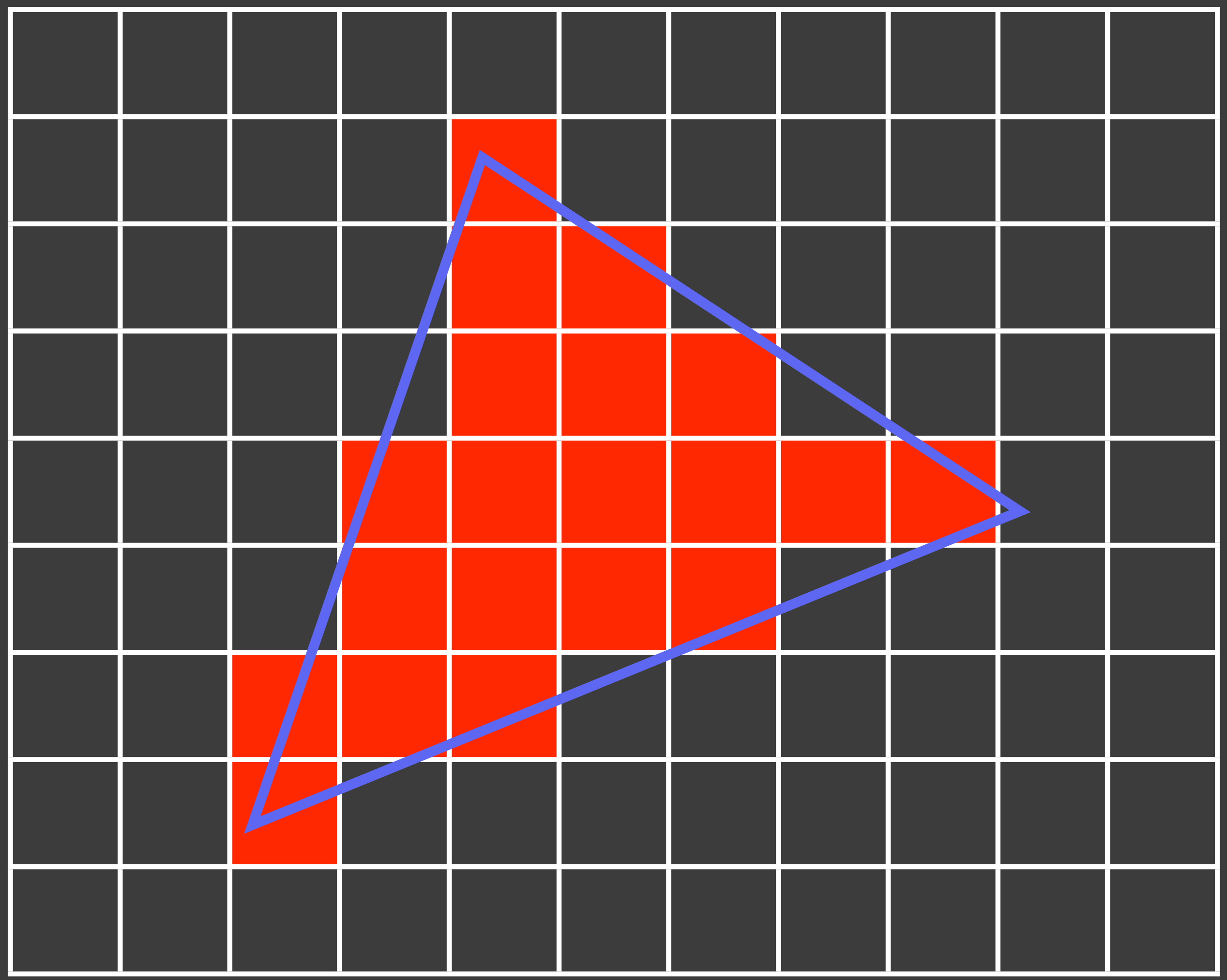
Scan Conversion

Discretize a polygon in image space to the image samples (pixels)

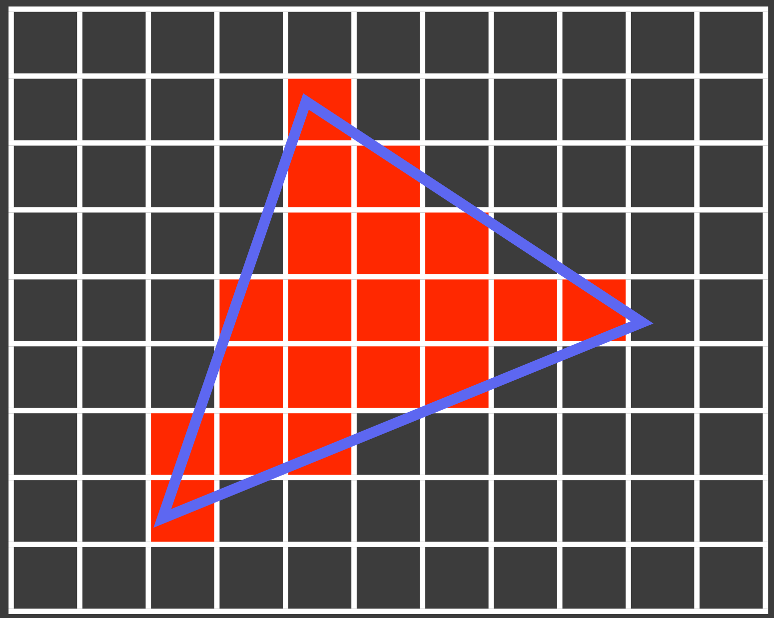


Scan Conversion

Discretize a polygon in image space to the image samples (pixels)

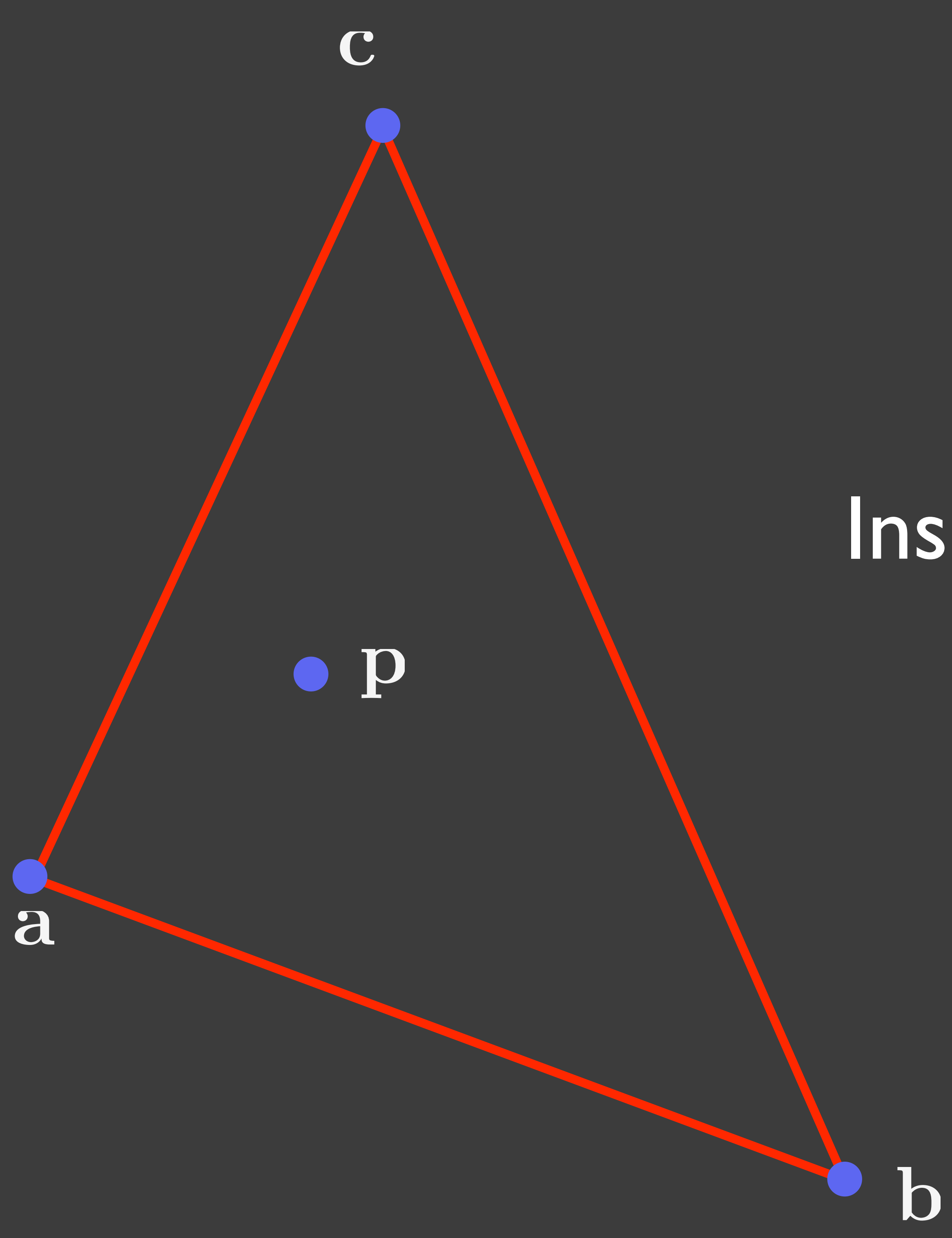


Triangle Scan Conversion



```
for each pixel (x, y):
  if (x, y) inside triangle:
    image(x, y) = compute color
```

Barycentric Coordinates

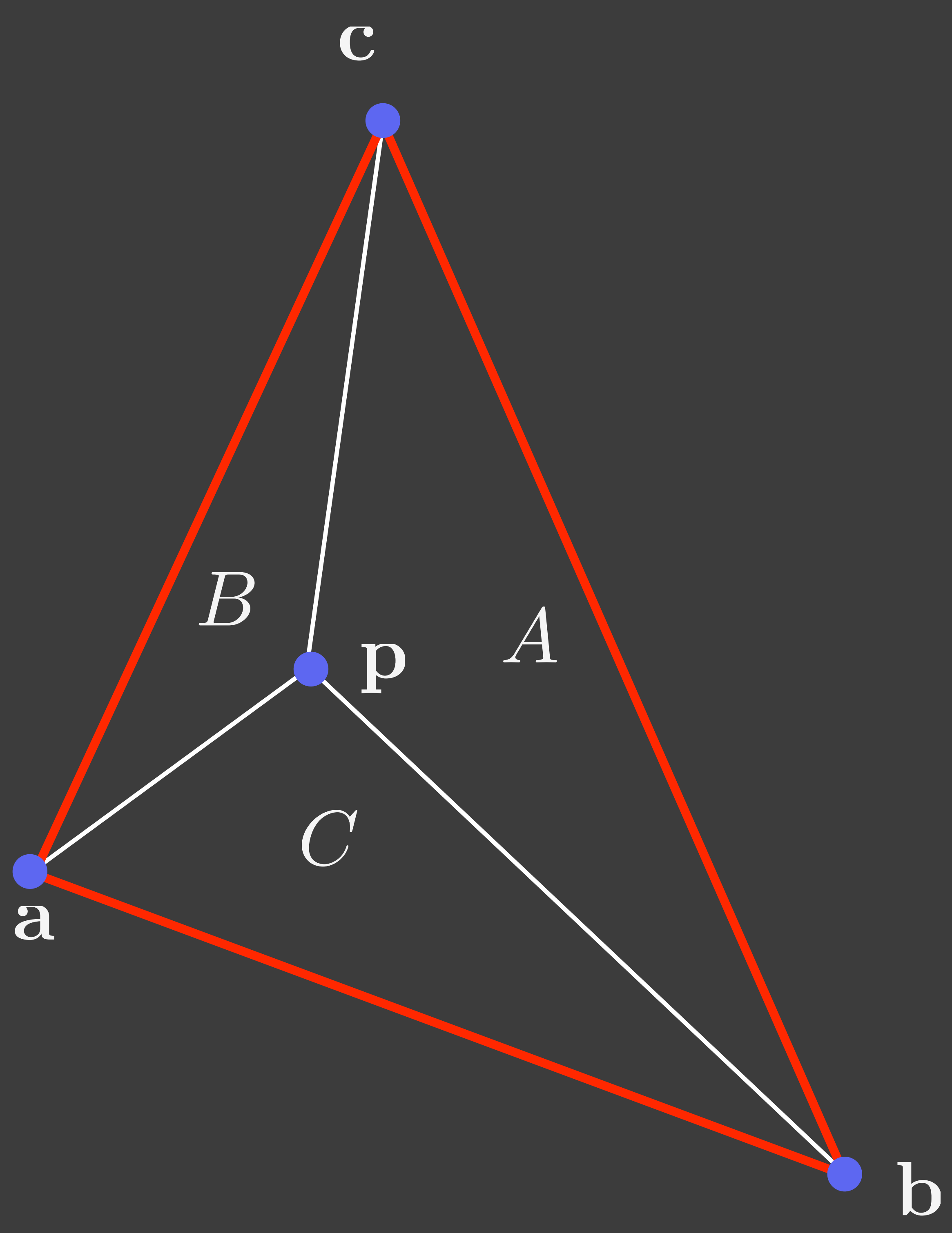


$$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$
$$\alpha + \beta + \gamma = 1$$

Inside test:

$$\alpha \geq 0, \beta \geq 0, \gamma \geq 0$$

Computing Barycentric Coordinates



$$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$
$$\alpha + \beta + \gamma = 1$$

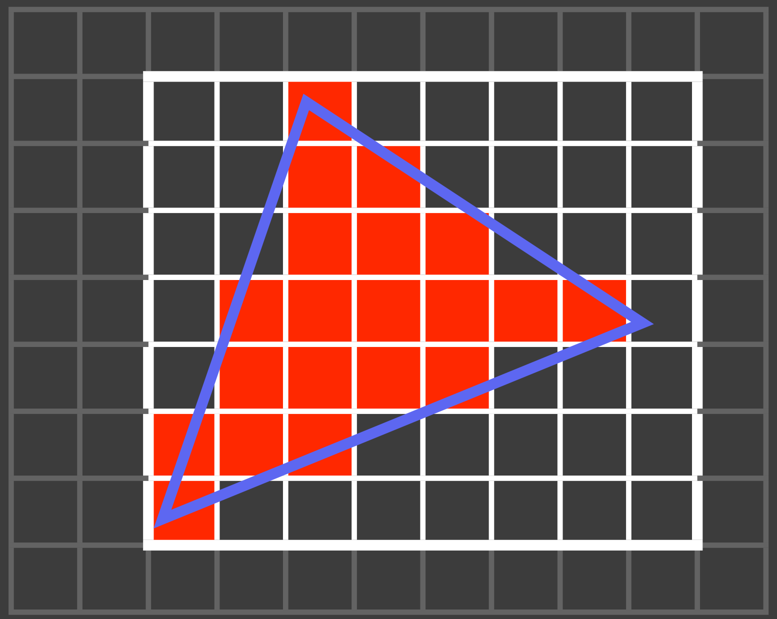
$$\alpha = \frac{area(A)}{area(\Delta)}$$

$$\beta = \frac{area(B)}{area(\Delta)}$$

$$\gamma = \frac{area(C)}{area(\Delta)}$$

Triangle Scan Conversion

Efficiency: Bound the triangle.



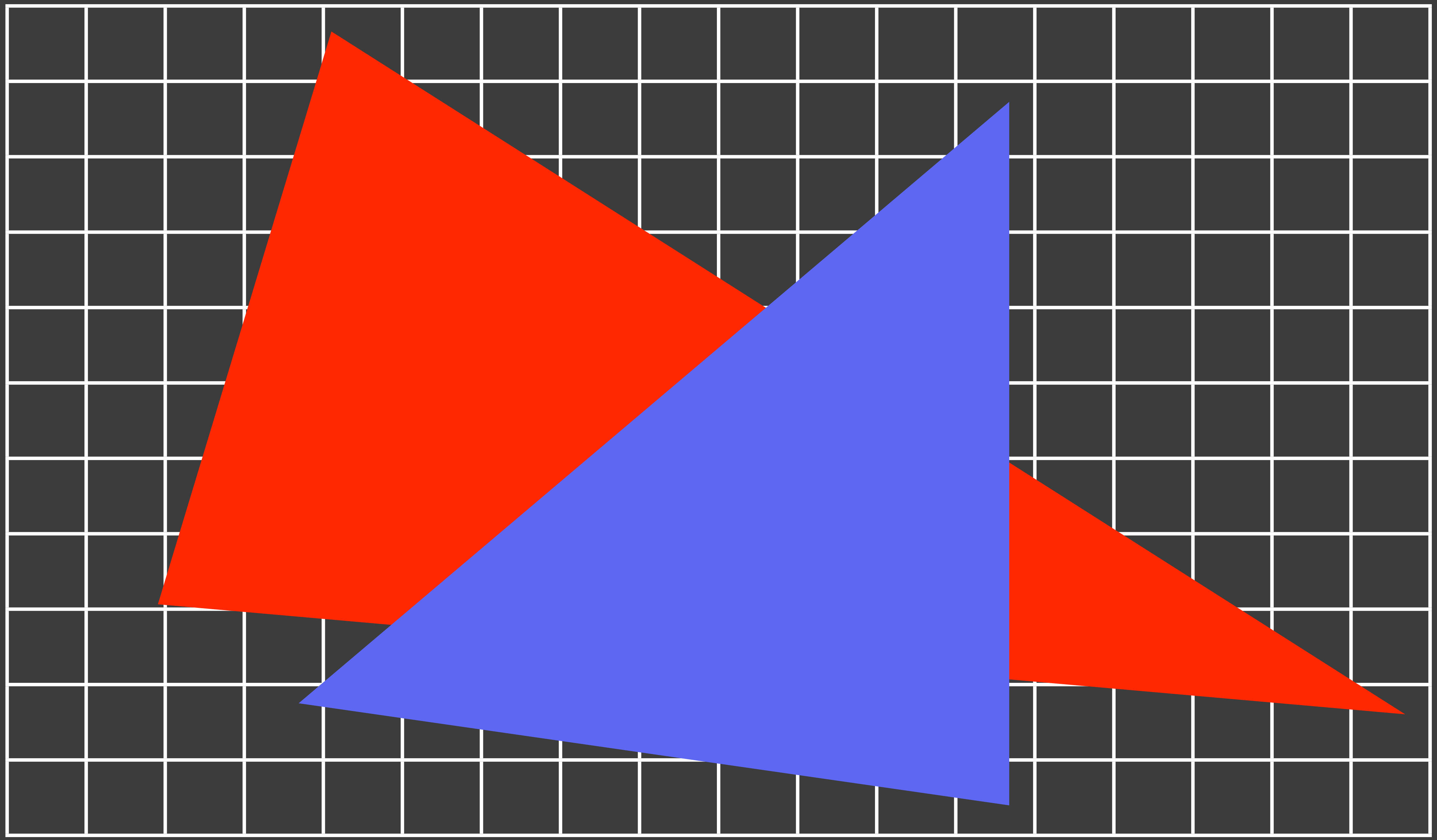
```
for y in [y_min, y_max]:
  for x in [x_min, x_max]:
    if (x, y) inside triangle:
      image(x, y) = compute color
```

There are even more efficient approaches.

There are algorithms for general polygons.

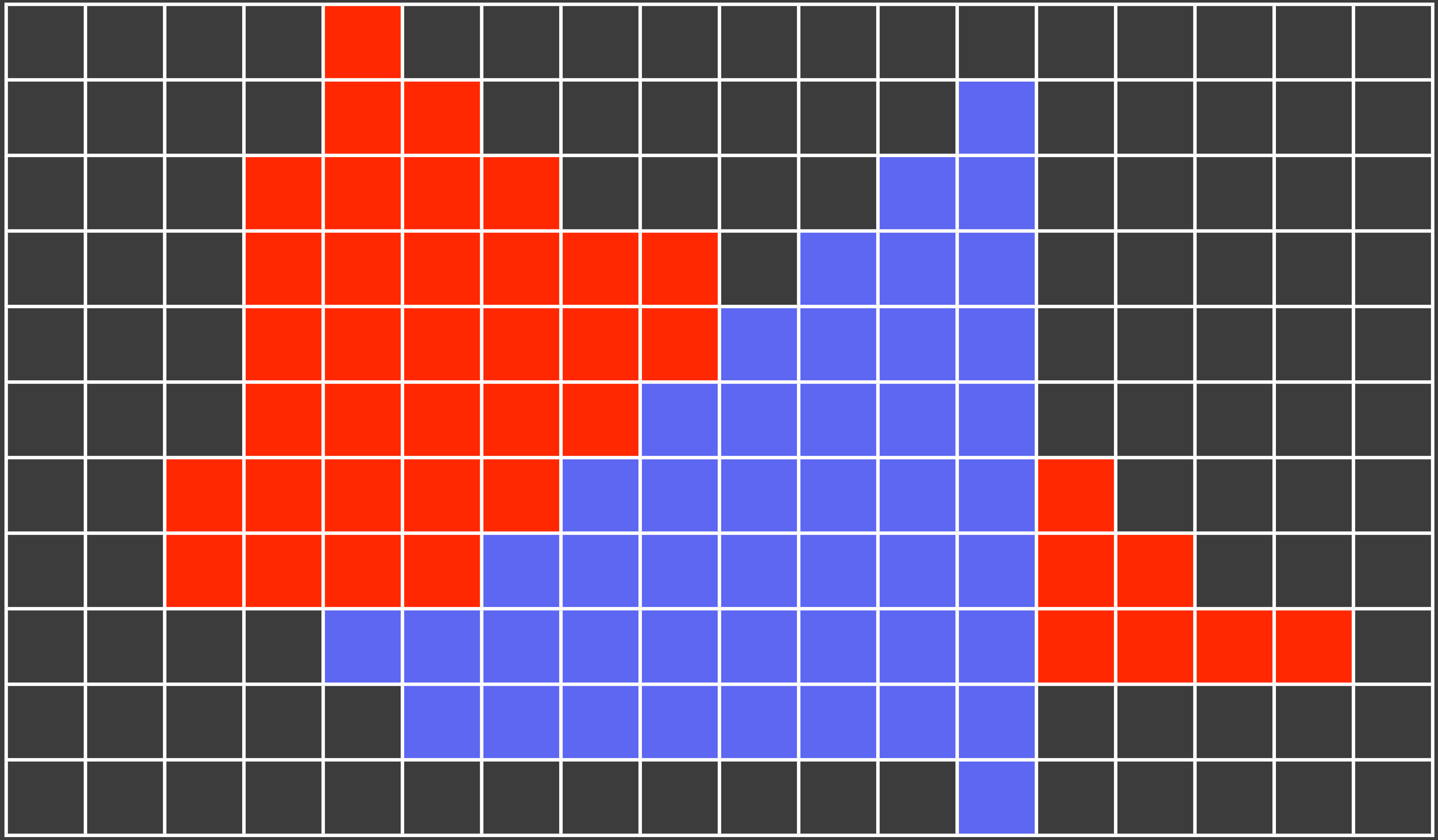
Z-Buffer

Handle shape occlusion during scan conversion.

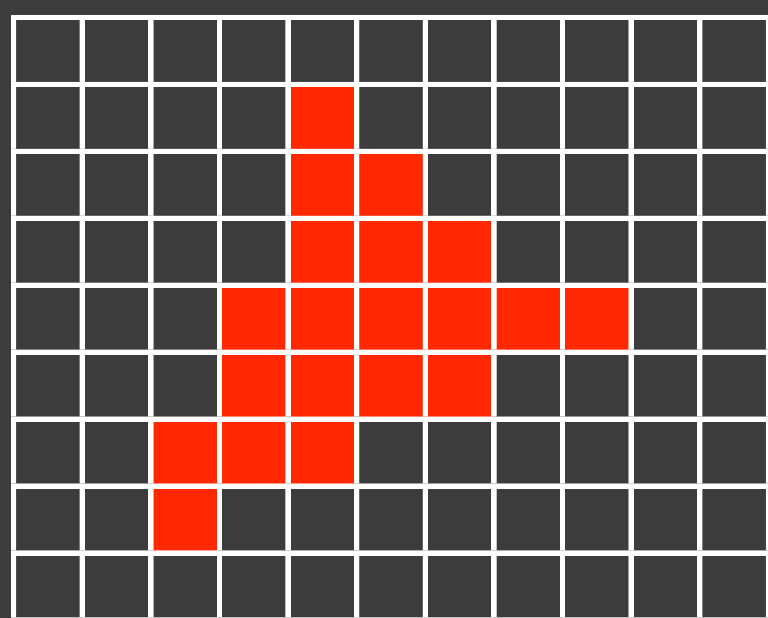


Z-Buffer

Handle shape occlusion during scan conversion.

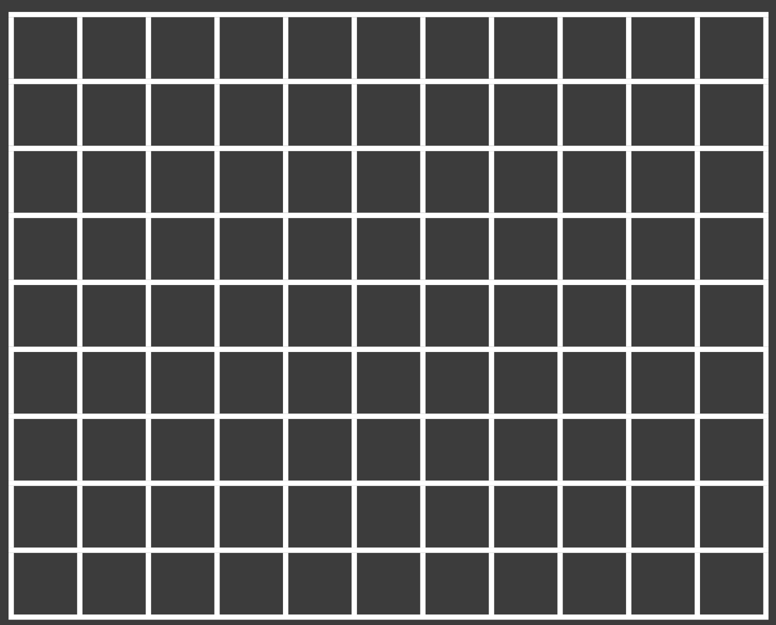


Z-Buffer



Array of Color

Initialize to Black



Array of z

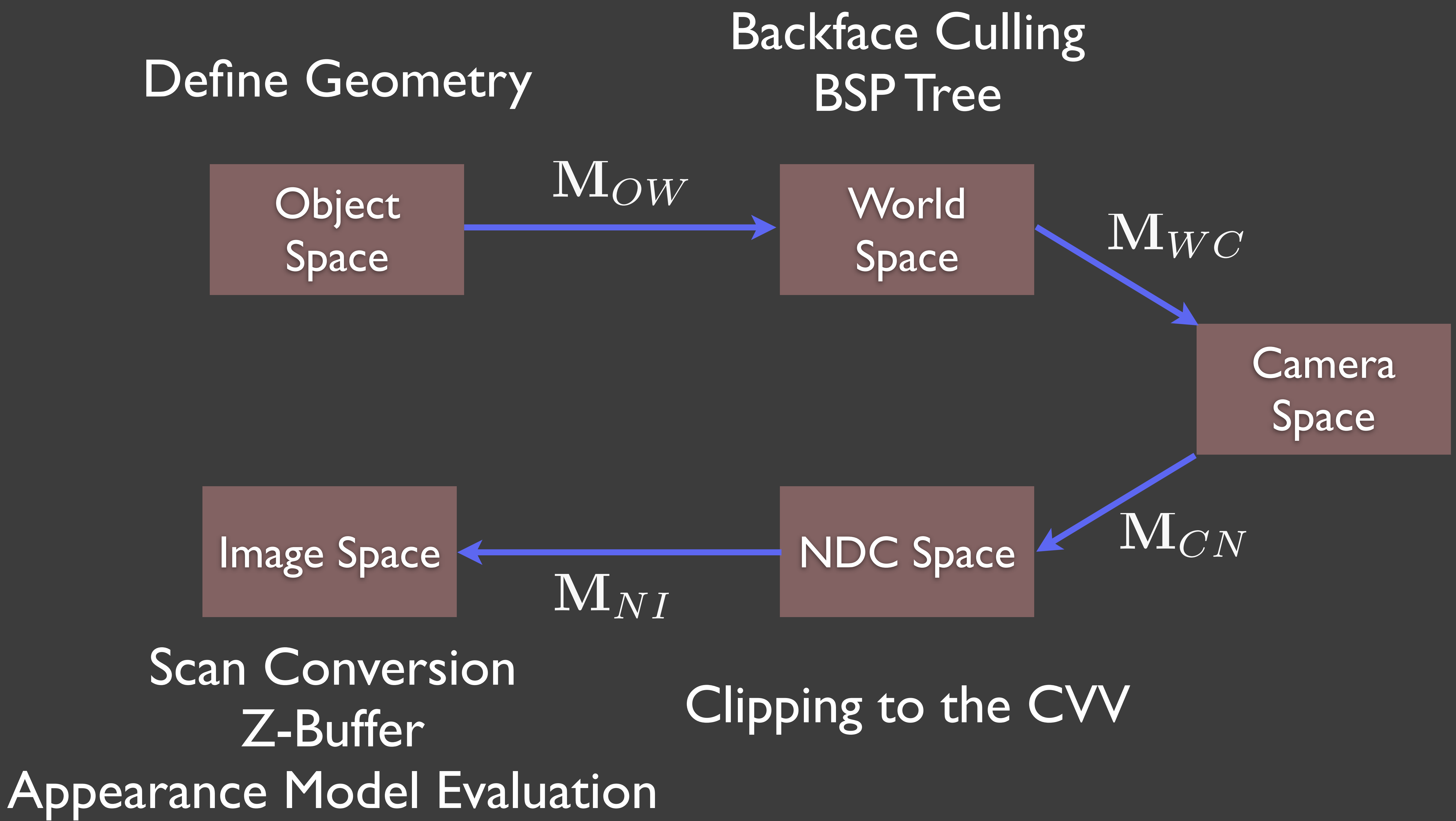
Initialize to ∞

Scan Conversion with the Z-Buffer

```
for each pixel (x, y):
  if (x, y) inside triangle:
    z = z(x, y)
    if z < zBuffer(x, y):
      zBuffer(x, y) = z
      image(x, y) = compute color
```

?

Rasterization: All Together



Rasterization	Ray Tracing
Efficient	Slow
Complicated	Much Simpler
Ideas show up in many other places	Ideas show up in many other places

Ray Tracing

```
for each pixel:
  for each object:
    if the object is visible in the pixel:
      compute the color and draw it
```

Ray Tracing

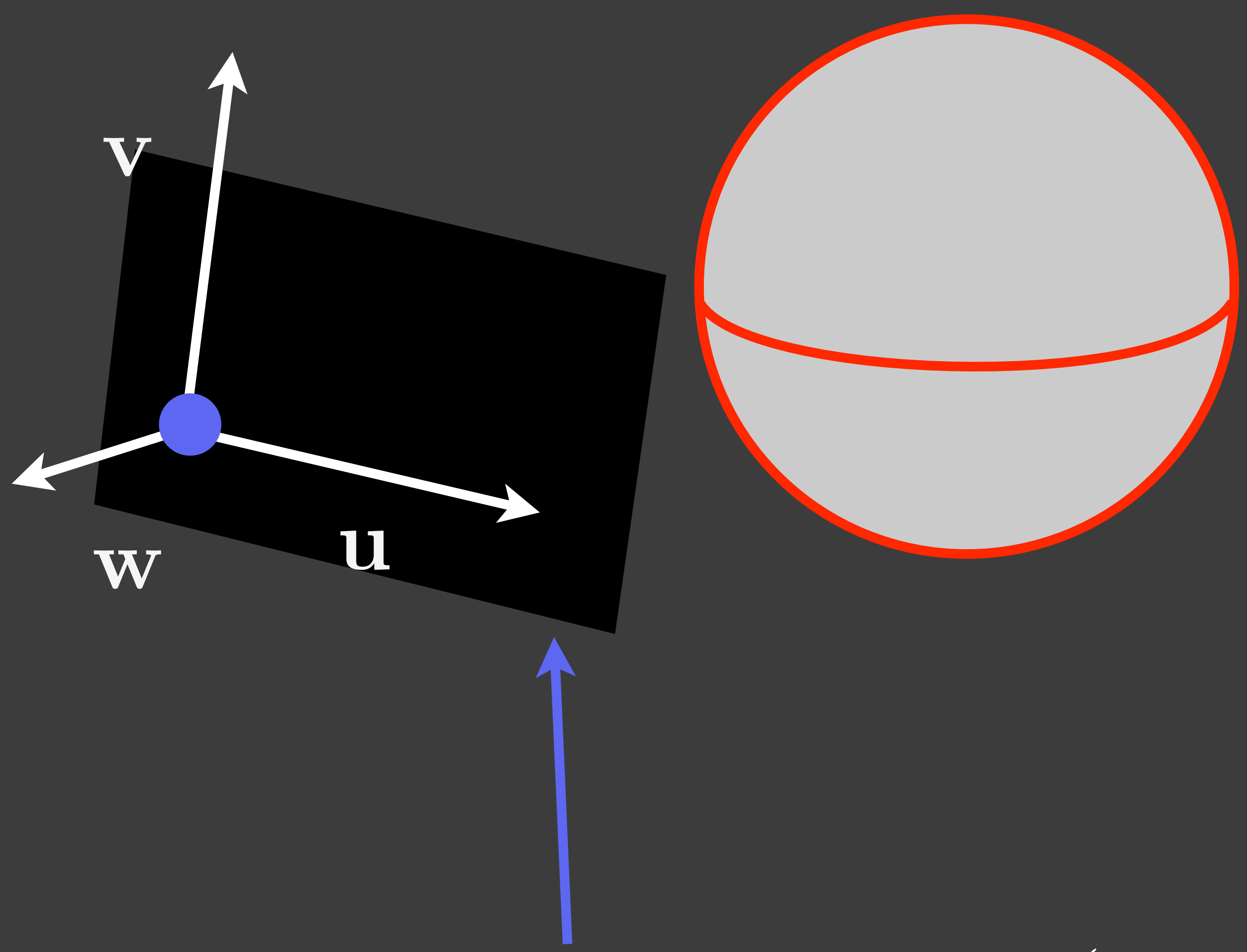
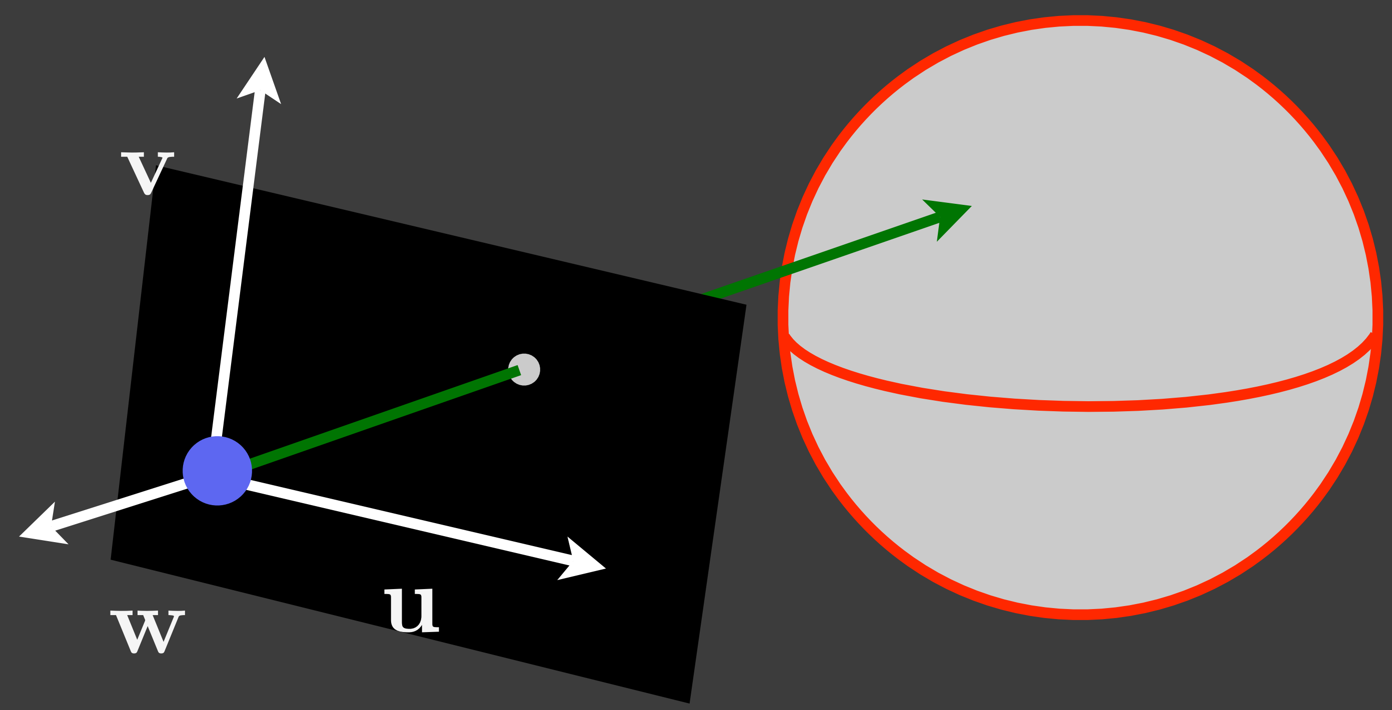
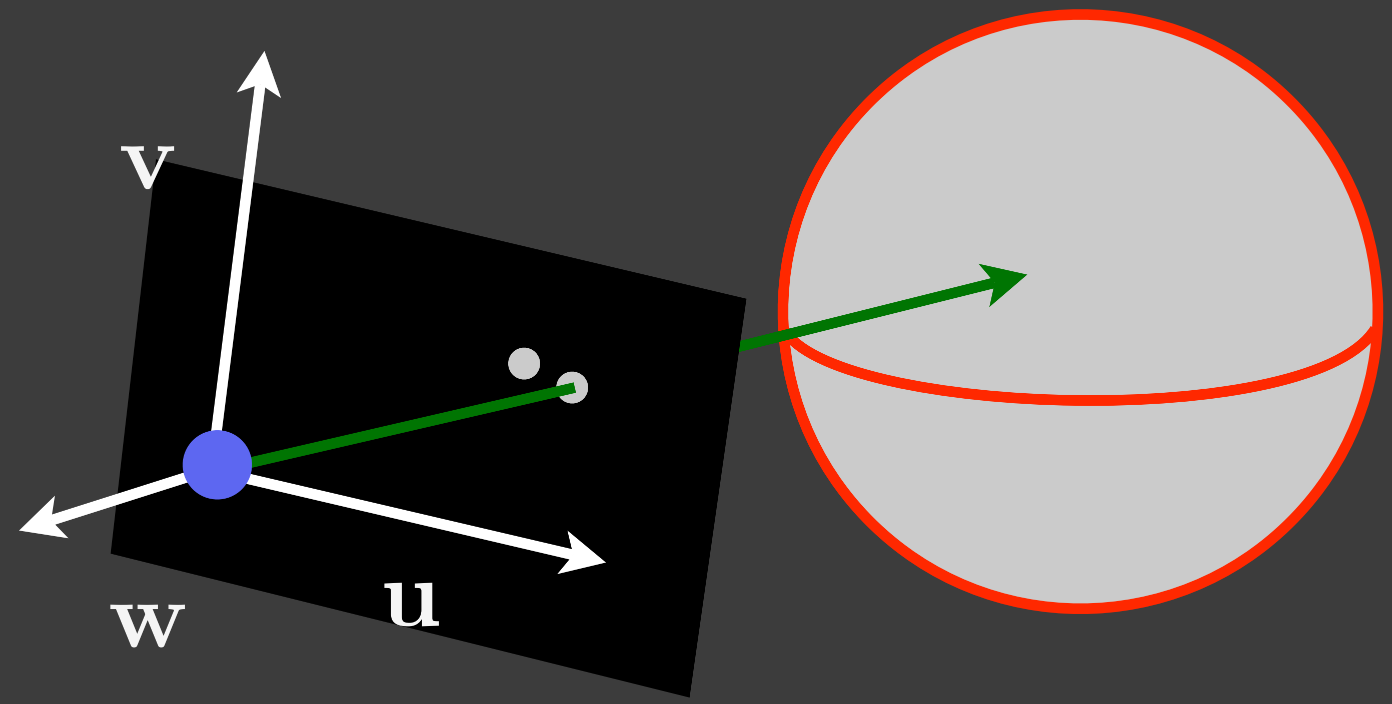


Image parameterized by $(\mathbf{u}, \mathbf{v})$

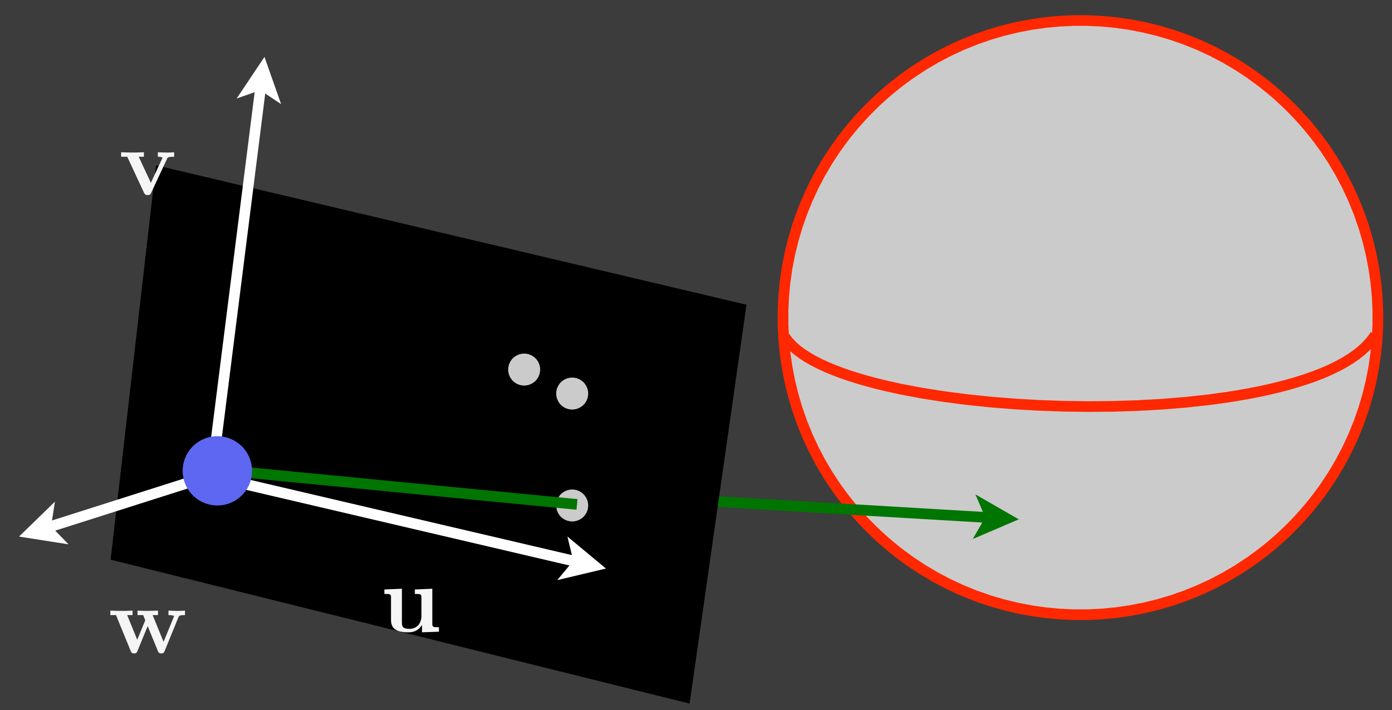
Ray Tracing



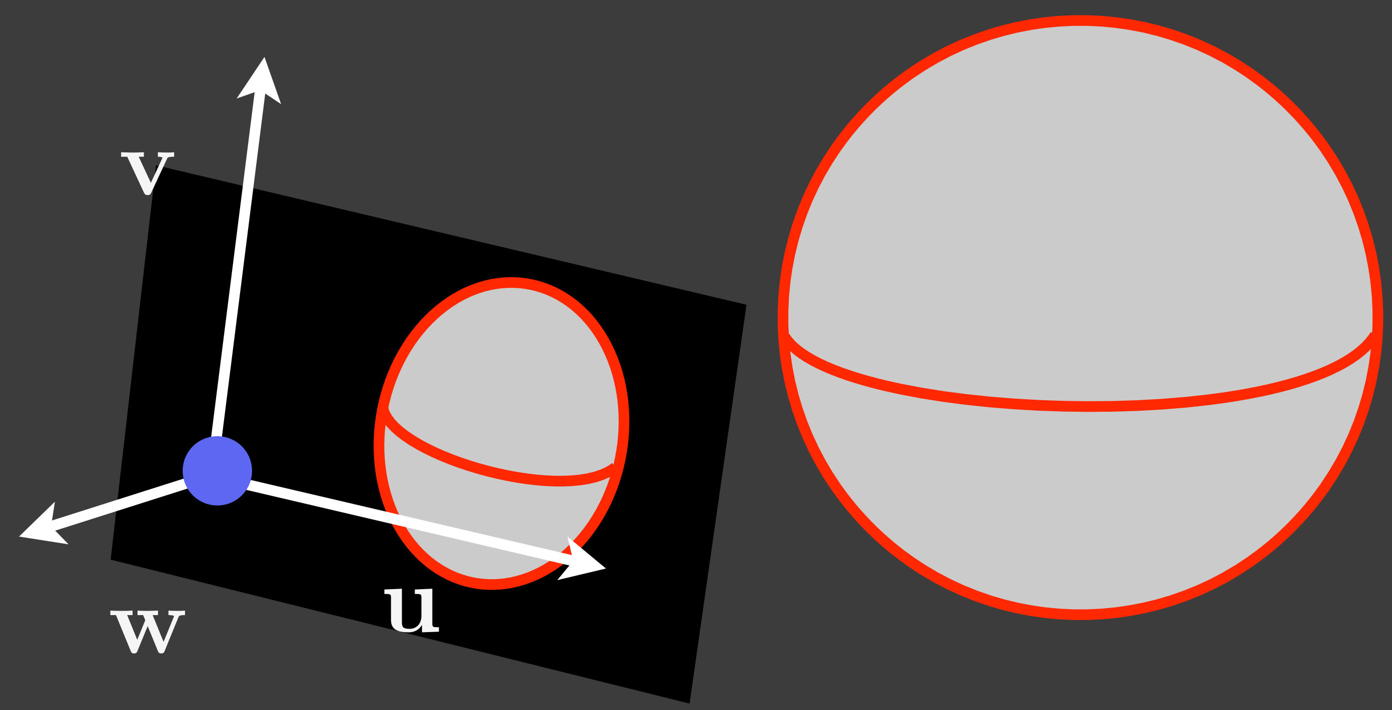
Ray Tracing



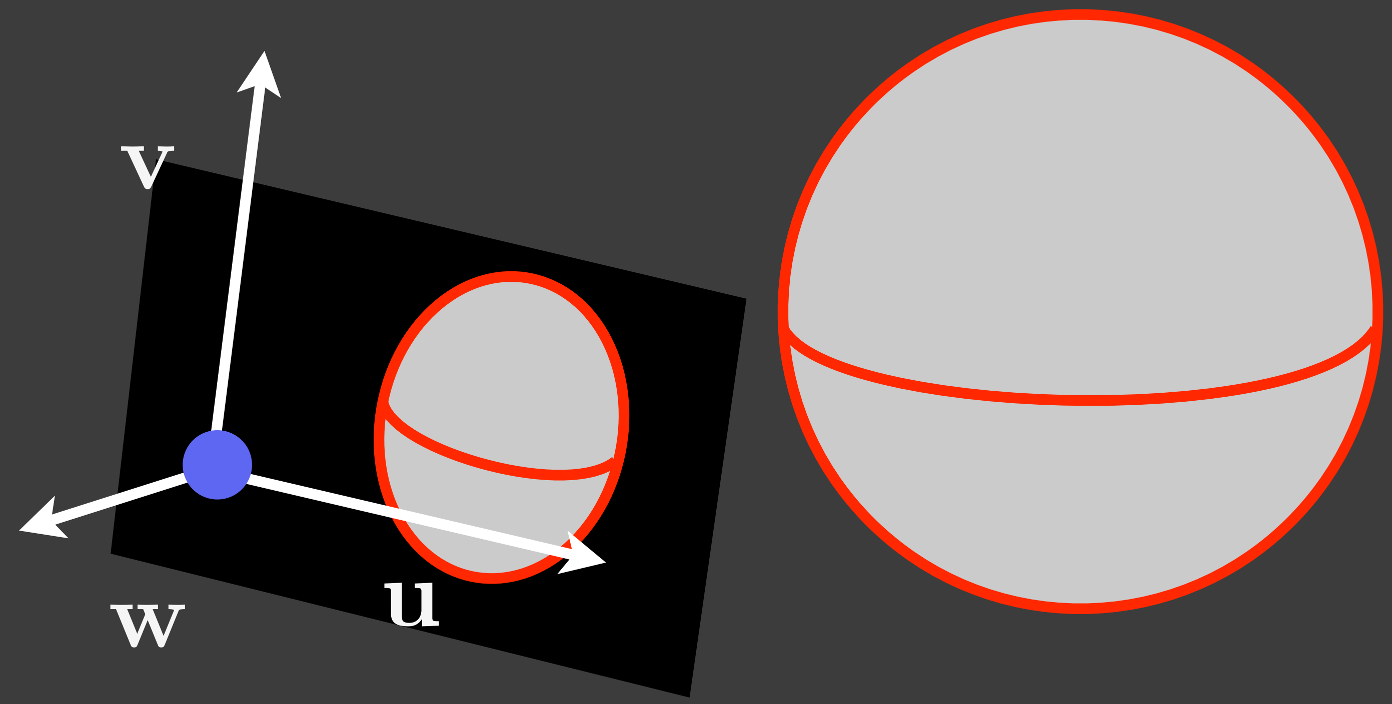
Ray Tracing



Ray Tracing



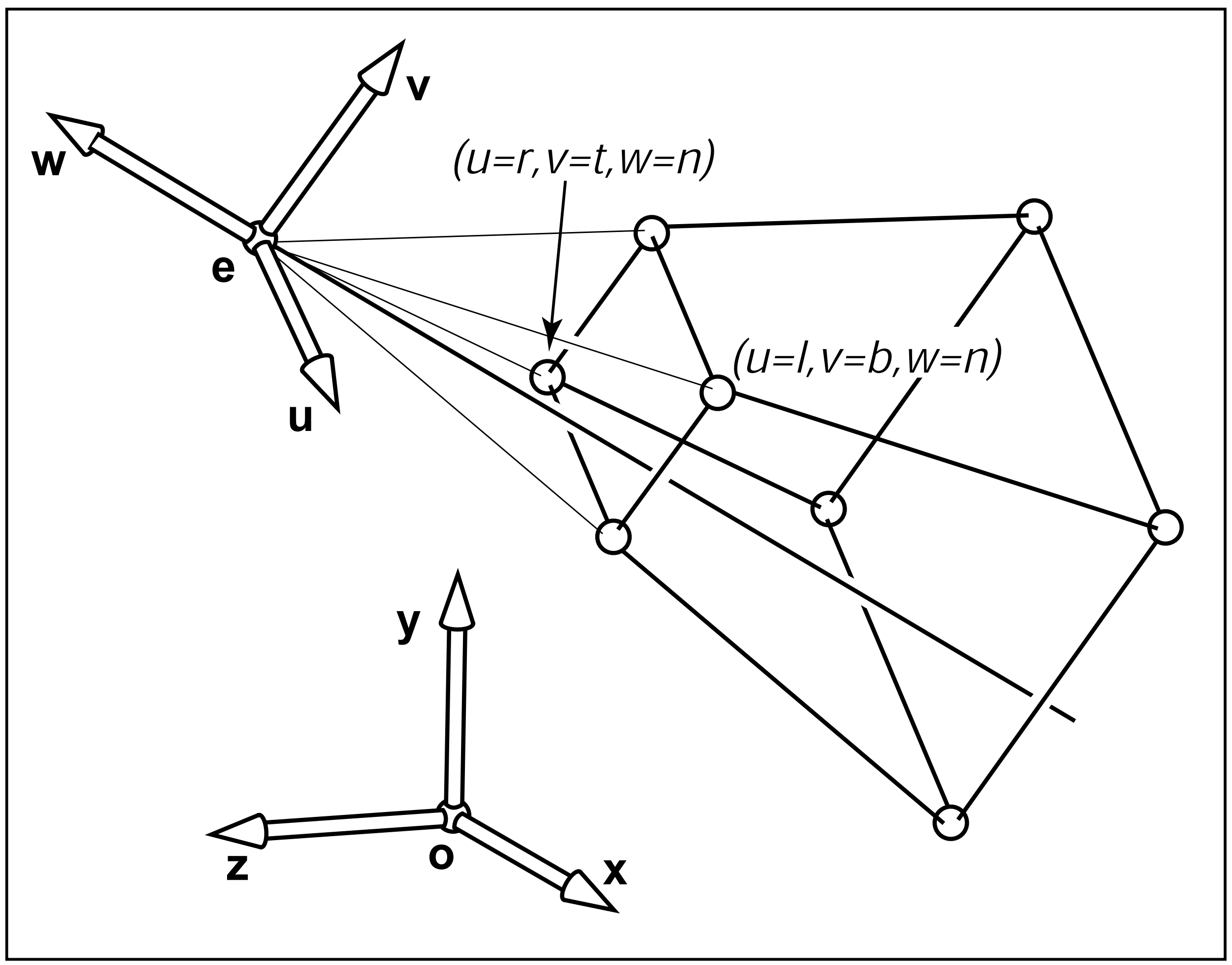
Ray Tracing

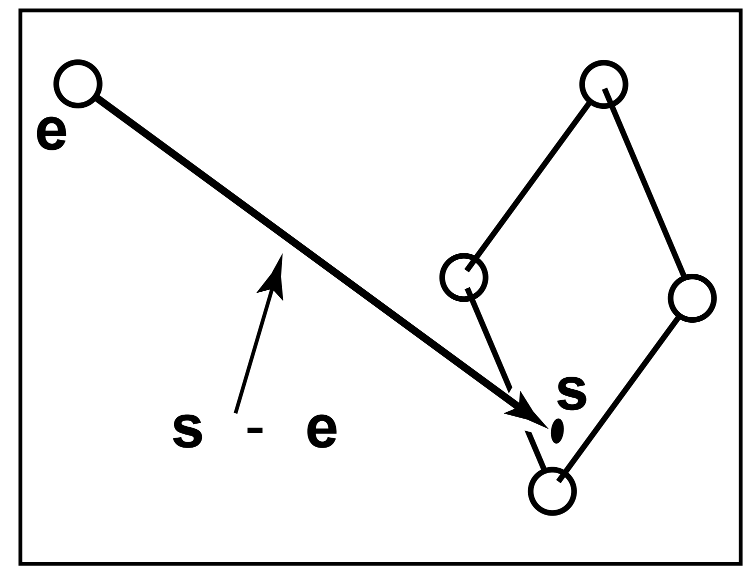


Questions:

- Constructing Rays
- Calculating intersections
- Determining the closest intersection
- Transforming objects

Ray construction





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

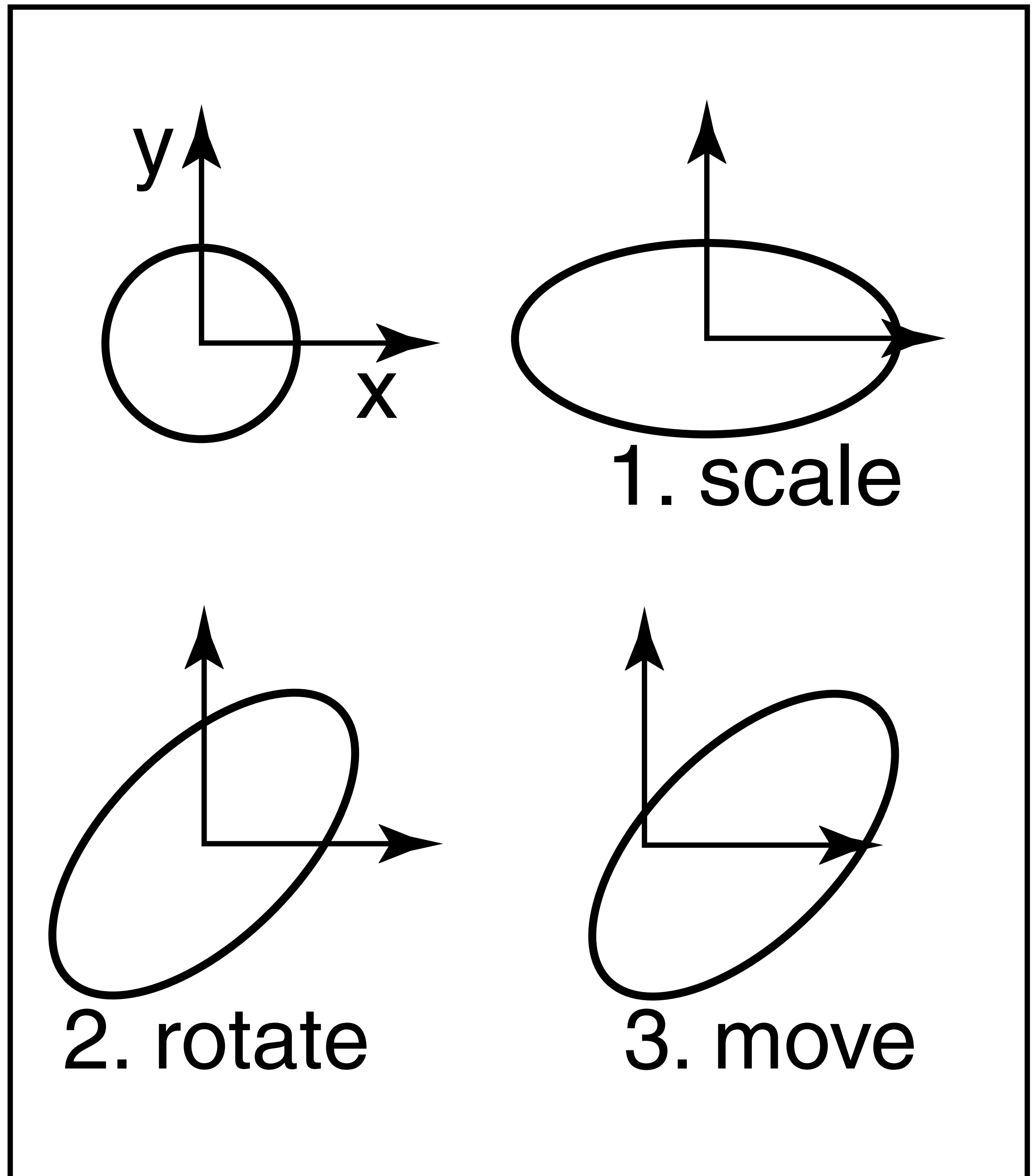
Implicit surface

- ray: $\mathbf{p}(t) = \mathbf{e} + t(\mathbf{s} - \mathbf{e})$
- surface: $f(\mathbf{p}) = 0$
- plug in value
  $f(\mathbf{p}) = 0$
  $\Rightarrow f(\mathbf{e} + t(\mathbf{s} - \mathbf{e})) = 0$
- solve for t
- if solution exists then ray hits and plugging the value back in $\mathbf{p}(t)$ will give the point of intersection

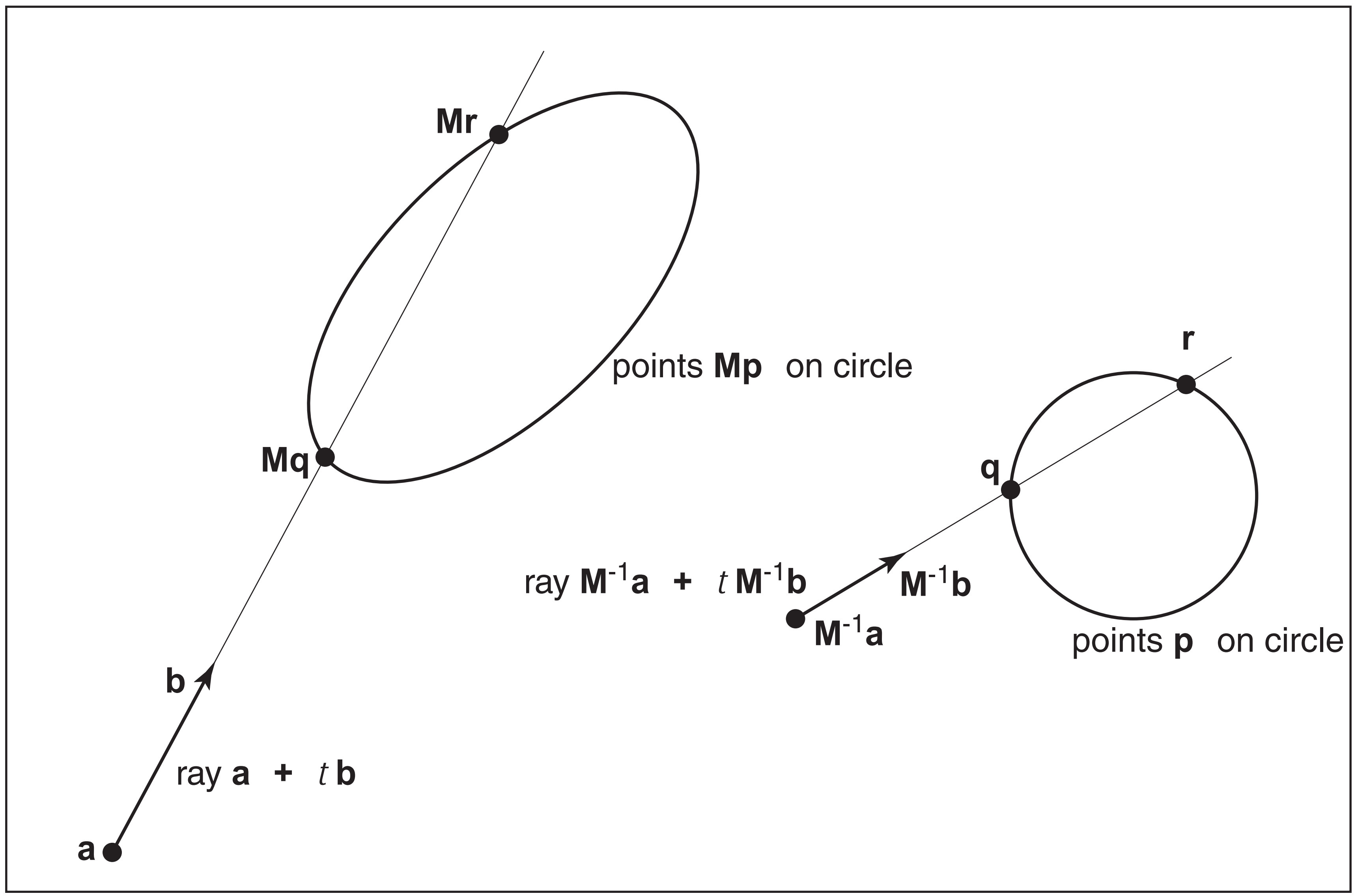
Parametric surface

- $\mathbf{e} = (e_x, e_y, e_z)$
- $\mathbf{d} = \mathbf{s} - \mathbf{e} = (d_x, d_y, d_z)$
- $f(u,v) = (f_x(u,v), f_y(u,v), f_z(u,v))$
- System of 3 equations in 3 unknowns:
  $e_x + td_x = f_x(u,v)$
  $e_y + td_y = f_y(u,v)$
  $e_z + td_z = f_z(u,v)$
- Solve for t, u, v. If solution exists, ray hits and plugging values back gives point of intersection.

Instancing



Instancing



Mr
points **Mp** on circle
Mq
r
q
ray $\mathbf{M}^{-1}\mathbf{a} + t\,\mathbf{M}^{-1}\mathbf{b}$
$\mathbf{M}^{-1}\mathbf{b}$
$\mathbf{M}^{-1}\mathbf{a}$
points **p** on circle
b
ray **a** + *t* **b**
a

Instancing

instance::hit(ray $\mathbf{a} + t\mathbf{b}$, real t_0, real t_1, hit-record rec)
ray $\mathbf{r'} = M^{-1}\mathbf{a} + tM^{-1}\mathbf{b}$
if (base-object→hit($\mathbf{r'}$, t_0, t_1, rec)) **then**
    rec.$\mathbf{n} = (M^{-1})^T$rec.$\mathbf{n}$
    return true
else
    return false