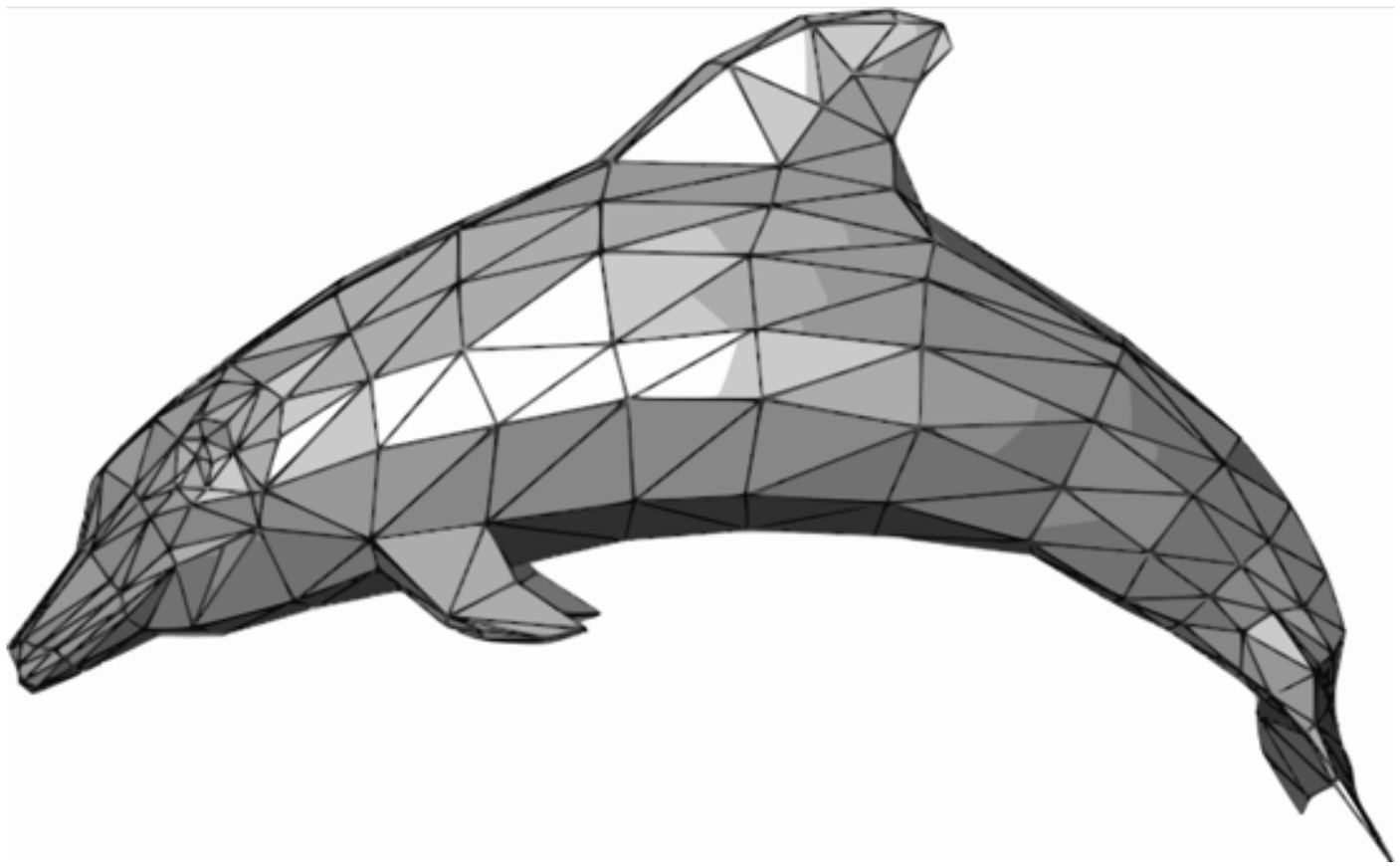
Triangle Meshes & Acceleration Structures

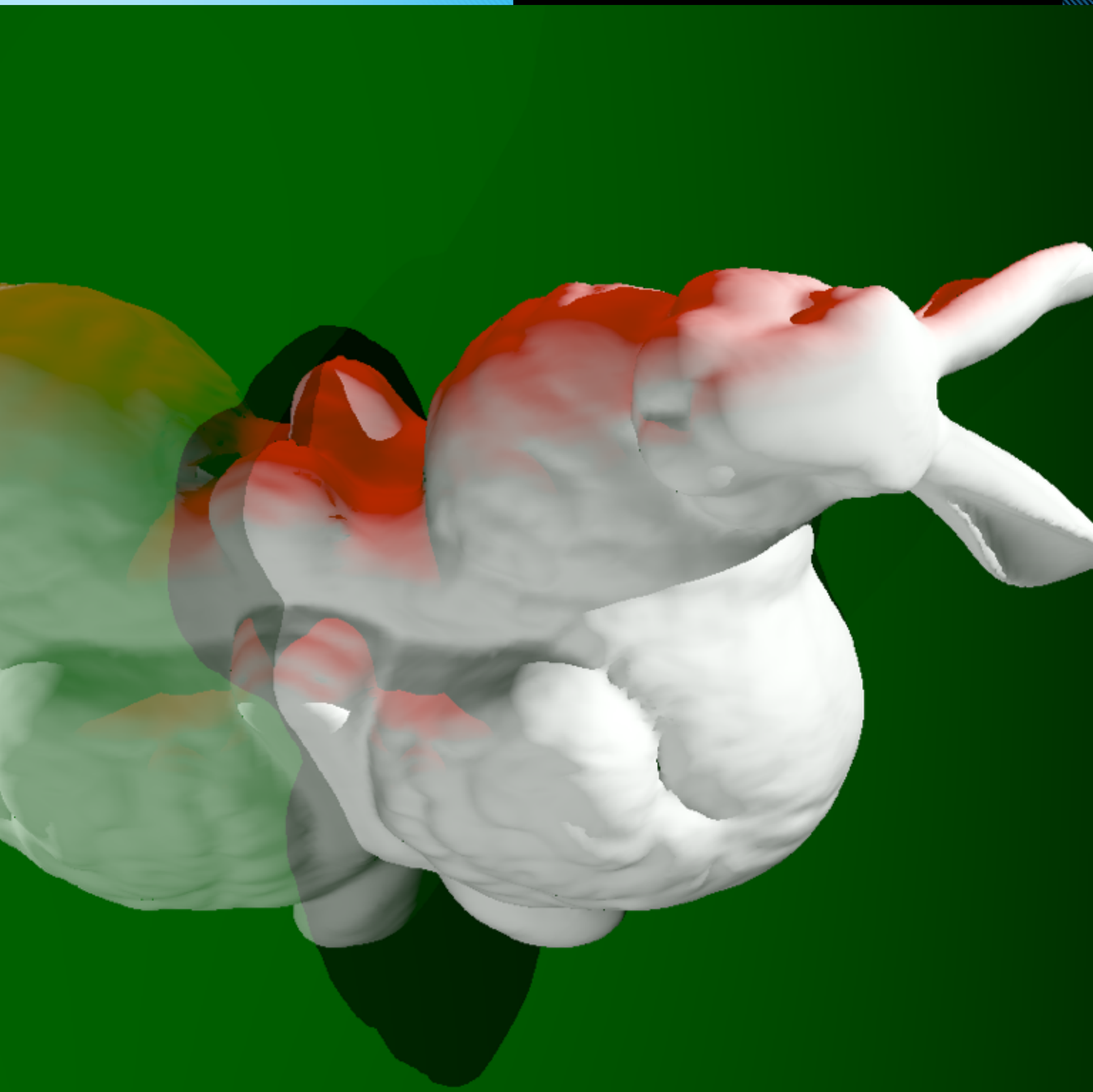


Overview

1. Triangle Meshes
2. Triangle Hit Function
3. PLY Files
4. Flat vs. Smooth Shading
5. Acceleration Structures

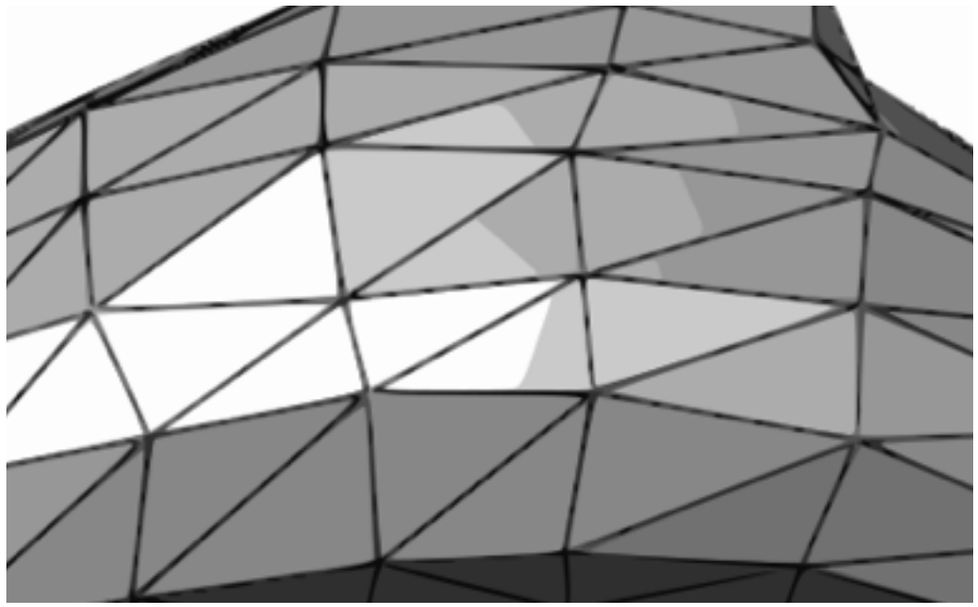
Triangle Meshes

- Implicit surfaces are great for modelling shapes exactly
- Triangle meshes excel at modelling complex shapes
  - For example: animals, trees, etc
  - But: spheres are only approximations of spheres



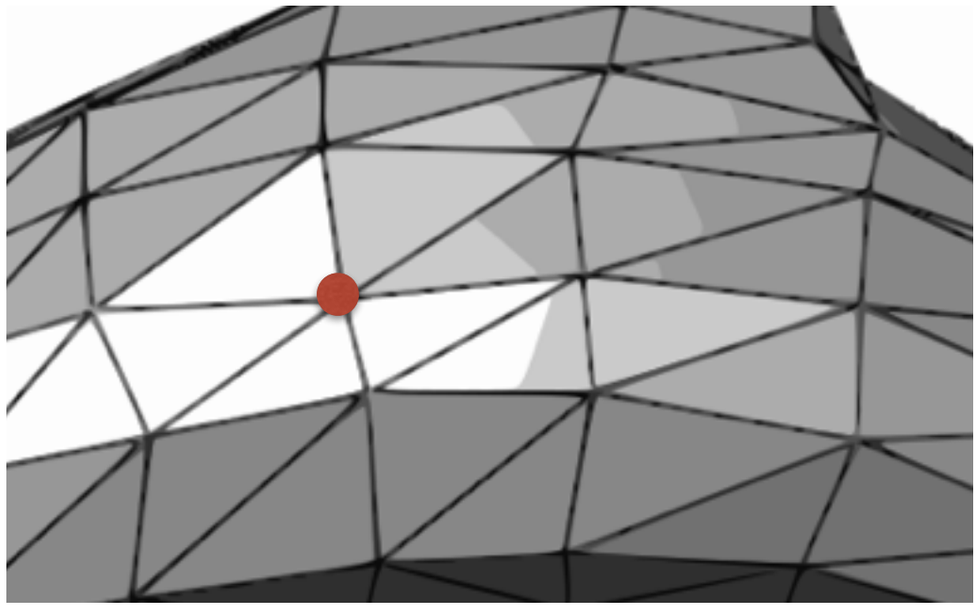
What are triangle meshes?

- Vertexes connected by edges
- Edges form triangles
- Vertexes may belong to many triangles



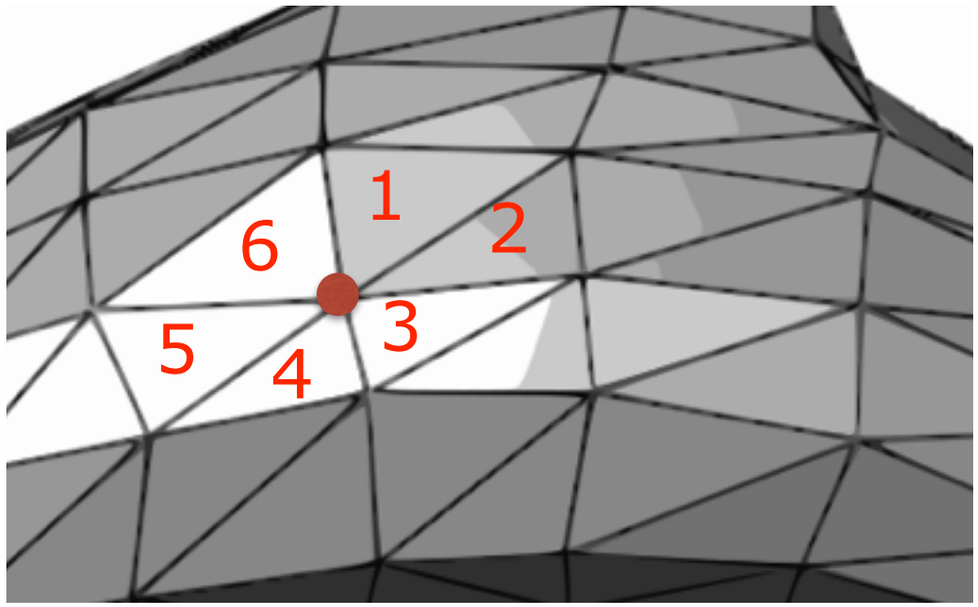
What are triangle meshes?

- Vertexes connected by edges
- Edges form triangles
- Vertexes may belong to many triangles



What are triangle meshes?

- Vertexes connected by edges
- Edges form triangles
- Vertexes may belong to many triangles



Compact Representation

Triangles

	P_1	P_2	P_3
	3	2	0
	1	4	3
	5	1	7
	7	5	2
	8	1	2
	7	8	6

Vertices

	x	y	z
0	1.23	0.2	12.2
1	3.44	9.47	3.09
2	2.42	3.0	3.0
3	…	…	…
4	…	…	…
5	…	…	…
6	…	…	…
7	…	…	…
8	…	…	…

Compact Representation

Triangles

	P_1	P_2	P_3
	3	2	0
	1	4	3
	5	1	7
	7	5	2
	8	1	2
	7	8	6

Vertices

	x	y	z
0	1.23	0.2	12.2
1	3.44	9.47	3.09
2	2.42	3.0	3.0
3	…	…	…
4	…	…	…
5	…	…	…
6	…	…	…
7	…	…	…
8	…	…	…

Compact Representation

Triangles

	P_1	P_2	P_3
	3	2	0
	1	4	3
	5	1	7
	7	5	2
	8	1	2
	7	8	6

Vertices

	x	y	z
0	1.23	0.2	12.2
1	3.44	9.47	3.09
2	2.42	3.0	3.0
3	…	…	…
4	…	…	…
5	…	…	…
6	…	…	…
7	…	…	…
8	…	…	…

Compact Representation

Triangles

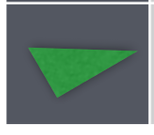
	P_1	P_2	P_3
	3	2	0
	1	4	3
	5	1	7
	7	5	2
	8	1	2
	7	8	6

Vertices

	x	y	z	normal	…
0	1.23	0.2	12.2		
1	3.44	9.47	3.09		
2	2.42	3.0	3.0		
3	…	…	…		
4	…	…	…		
5	…	…	…		
6	…	…	…		
7	…	…	…		
8	…	…	…		

The Hit Function

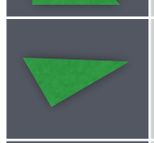
The Naive Way

	P_1	P_2	P_3
	3	2	0
	1	4	3
	5	1	7
	7	5	2
	8	1	2
	7	8	6

- Check for intersection with each triangle one-by-one
- find the intersection point closest to camera
- return the closest intersection point (if any)

The Hit Function

The Naive Way

	P_1	P_2	P_3
	3	2	0
	1	4	3
	5	1	7
	7	5	2
	8	1	2

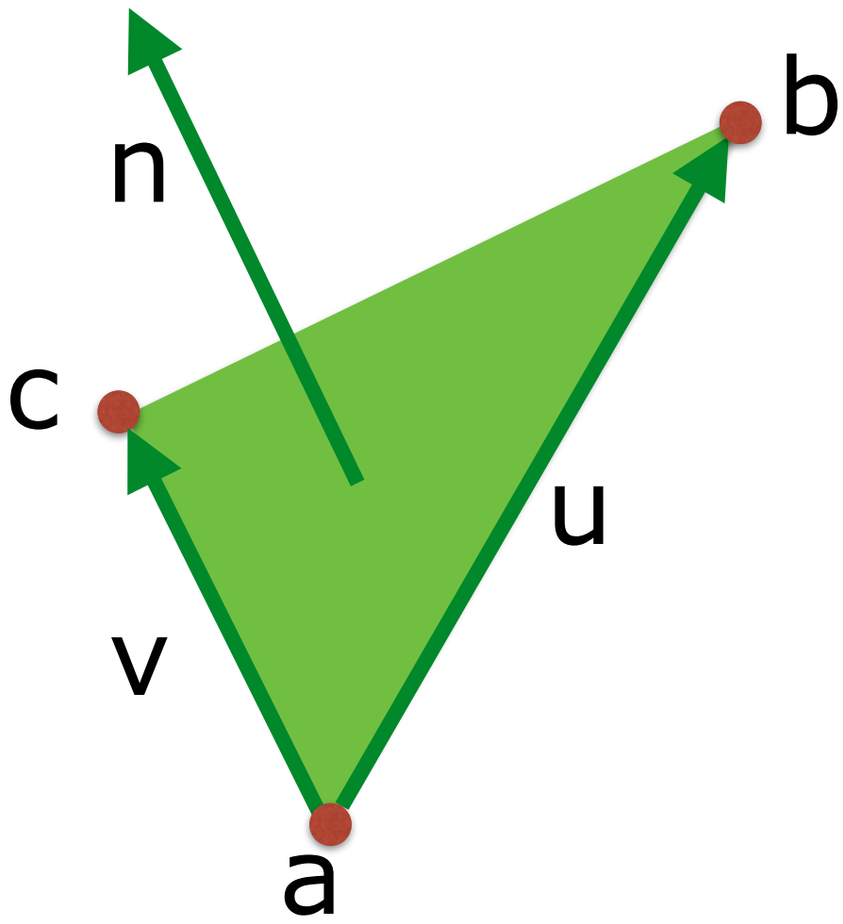
- Check for intersection with each triangle one-by-one
- find the intersection point closest to camera
- return the closest intersection point (if any)

This is not feasible for large triangle meshes!
More about acceleration structures later.

Triangle Hit Function

The Easy Part

Calculating the normal vector

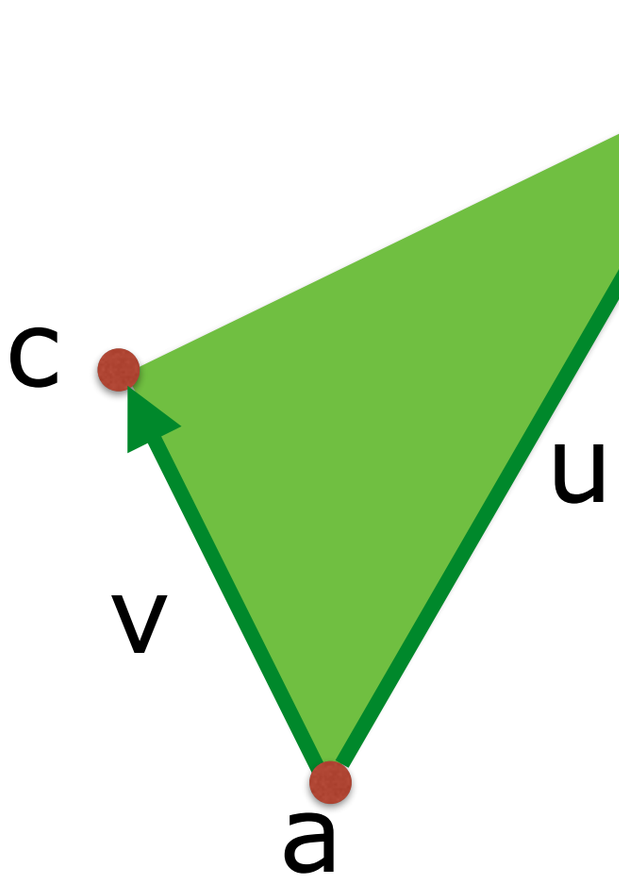


$$u = b - a$$

$$v = c - a$$

$$n = \frac{u \times v}{|u \times v|}$$

Testing for Intersection



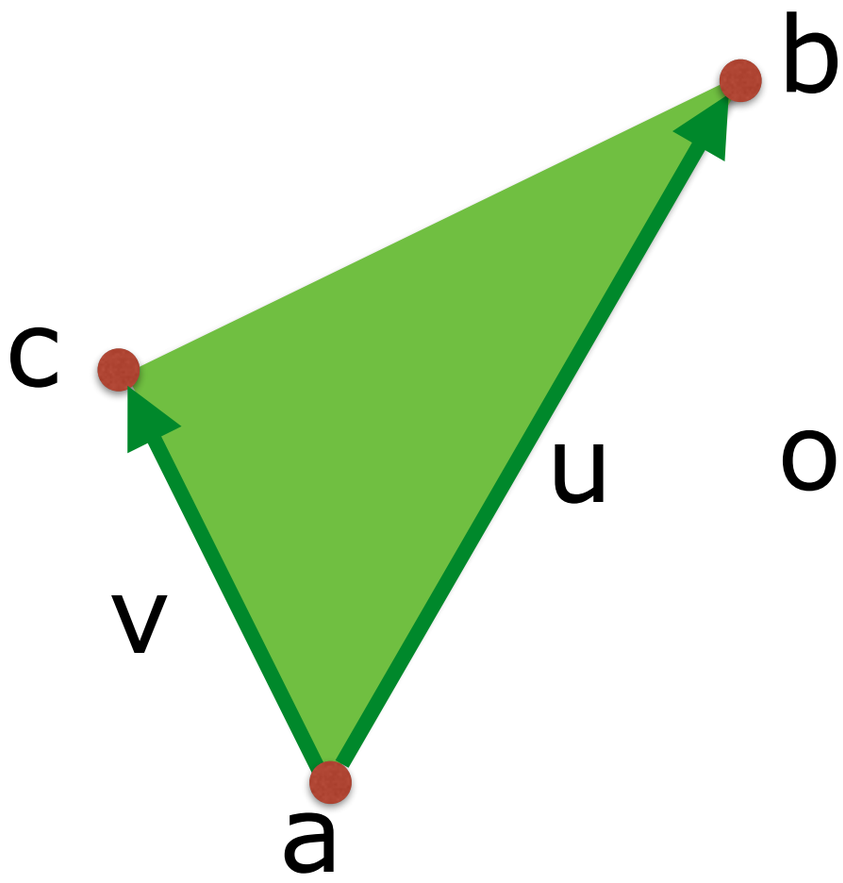
Equation for triangle

$$p = \alpha a + \beta b + \gamma c$$

$$\alpha + \beta + \gamma = 1$$

$$0 \leq \alpha, \beta, \gamma \leq 1$$

Testing for Intersection



Equation for triangle

$$p = \alpha a + \beta b + \gamma c$$

$$\alpha + \beta + \gamma = 1$$

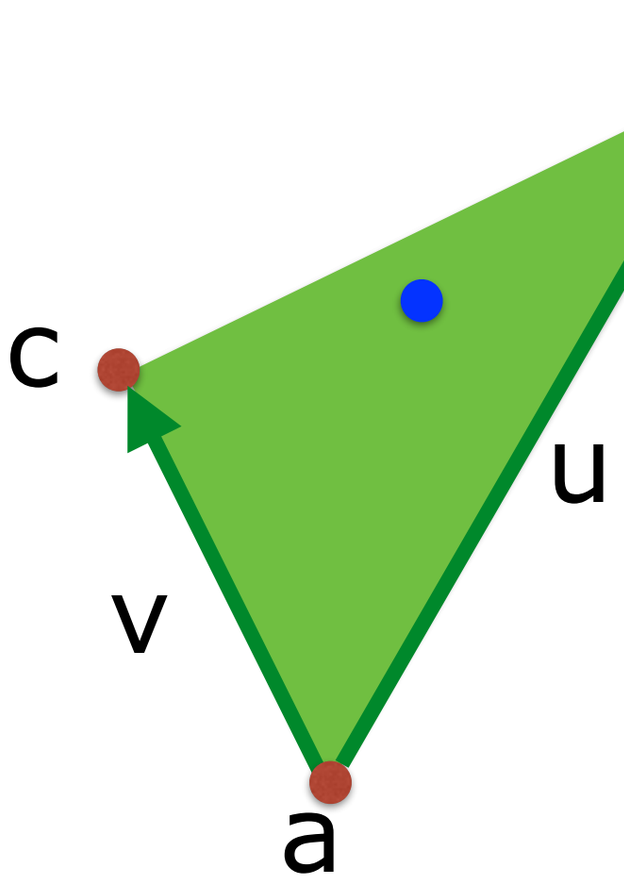
$$0 \le \alpha, \beta, \gamma \le 1$$

---

or: $$p = a + \beta u + \gamma v$$

$$0 \le \beta, \gamma, (\beta + \gamma) \le 1$$

Testing for Intersection



Equation for triangle

$$p = \alpha a + \beta b + \gamma c$$

$$\alpha + \beta + \gamma = 1$$

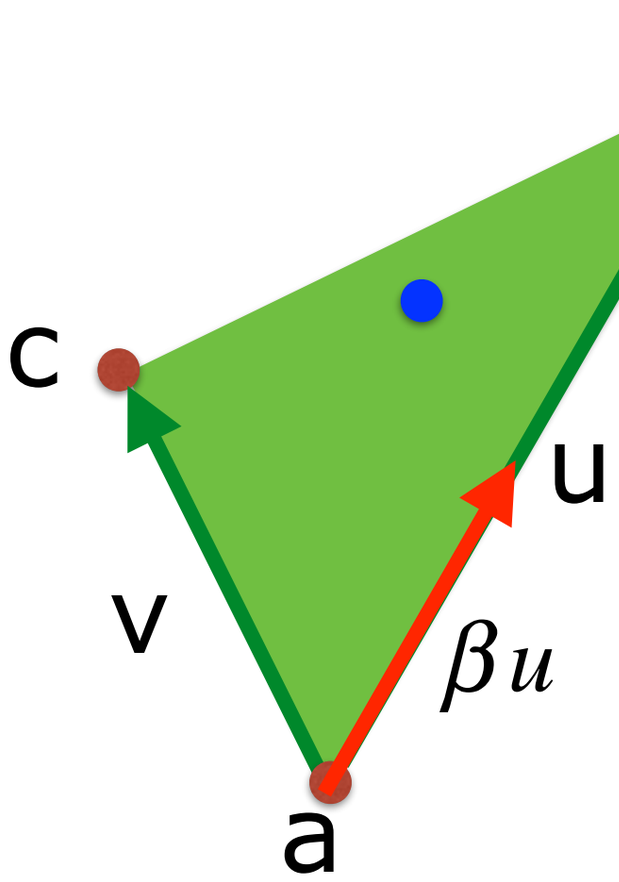
$$0 \le \alpha, \beta, \gamma \le 1$$

---

or: $p = a + \beta u + \gamma v$

$$0 \le \beta, \gamma, (\beta + \gamma) \le 1$$

Testing for Intersection



Equation for triangle

$$p = \alpha a + \beta b + \gamma c$$

$$\alpha + \beta + \gamma = 1$$

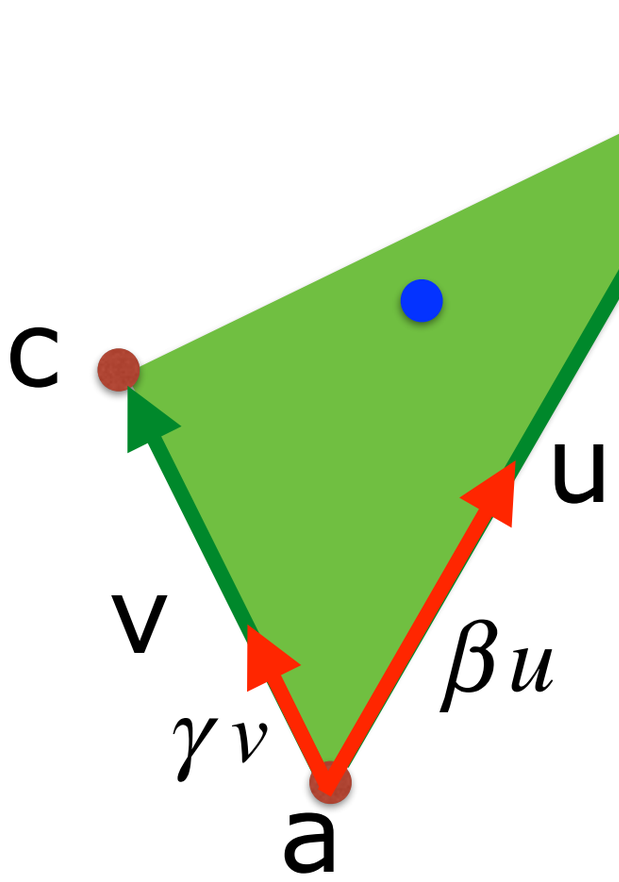
$$0 \le \alpha, \beta, \gamma \le 1$$

---

or: $p = a + \beta u + \gamma v$

$$0 \le \beta, \gamma, (\beta + \gamma) \le 1$$

Testing for Intersection



Equation for triangle

$$p = \alpha a + \beta b + \gamma c$$

$$\alpha + \beta + \gamma = 1$$

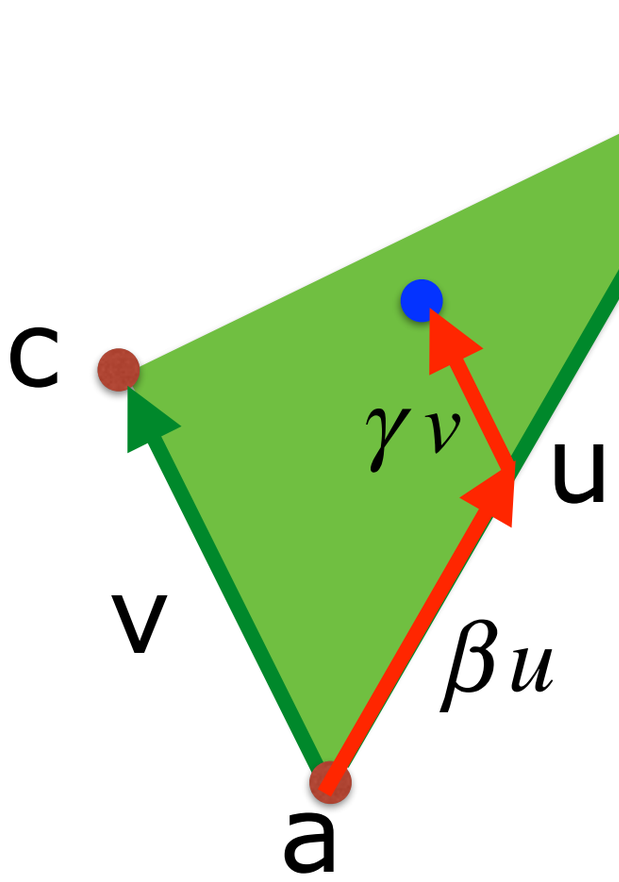
$$0 \le \alpha, \beta, \gamma \le 1$$

---

or: $p = a + \beta u + \gamma v$

$$0 \le \beta, \gamma, (\beta + \gamma) \le 1$$

Testing for Intersection



Equation for triangle

$$p = \alpha a + \beta b + \gamma c$$

$$\alpha + \beta + \gamma = 1$$

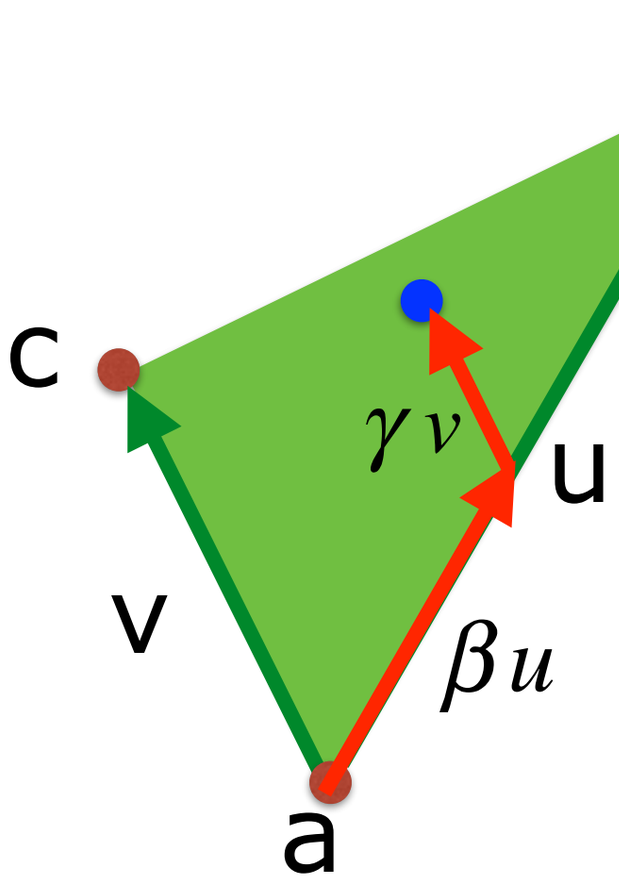
$$0 \le \alpha, \beta, \gamma \le 1$$

---

or: $p = a + \beta u + \gamma v$

$$0 \le \beta, \gamma, (\beta + \gamma) \le 1$$

Testing for Intersection



Equation for triangle

$$p = \alpha a + \beta b + \gamma c$$

$$\alpha + \beta + \gamma = 1$$

$$0 \le \alpha, \beta, \gamma \le 1$$

---

or: $p = a + \beta u + \gamma v$

$$0 \le \beta, \gamma, (\beta + \gamma) \le 1$$

Equation for ray

$$p = o + t\,d$$

Finding the intersection

$$p = a + \beta u + \gamma v \qquad u = b - a$$

$$p = o + t\,d \qquad v = c - a$$

Finding the intersection

$$p = a + \beta u + \gamma v \qquad u = b - a$$
$$p = o + t\,d \qquad v = c - a$$

$$o + t\,d = a + \beta(b - a) + \gamma(c - a)$$

Finding the intersection

$$p = a + \beta u + \gamma v \qquad u = b - a$$

$$p = o + t\,d \qquad v = c - a$$

$$o + t\,d = a + \beta(b-a) + \gamma(c-a)$$

$$\beta(a-b) + \gamma(a-c) + t\,d = a - o$$

Finding the intersection

$$p = a + \beta u + \gamma v \qquad u = b - a$$
$$p = o + t\,d \qquad v = c - a$$

$$o + t\,d = a + \beta(b-a) + \gamma(c-a)$$

$$\beta(a-b) + \gamma(a-c) + t\,d = a - o$$

$$\beta(a_x - b_x) + \gamma(a_x - c_x) + t\,d_x = a_x - o_x$$
$$\beta(a_y - b_y) + \gamma(a_y - c_y) + t\,d_y = a_y - o_y$$
$$\beta(a_z - b_z) + \gamma(a_z - c_z) + t\,d_z = a_z - o_z$$

Finding the intersection

- 3 equations with 3 unknowns (β,γ,t)
- can be solved using Cramer's rule
- we only need t to get the intersection point
- but we need β and γ later (and α=1-β-γ, too)

$$\beta(a_x - b_x) + \gamma(a_x - c_x) + t\,d_x = a_x - o_x$$
$$\beta(a_y - b_y) + \gamma(a_y - c_y) + t\,d_y = a_y - o_y$$
$$\beta(a_z - b_z) + \gamma(a_z - c_z) + t\,d_z = a_z - o_z$$

Finding the intersection

- 3 equations with 3 unknowns (β,γ,t)
- can be solved using Cramer's rule
- we only need t to get the intersection point
- but we need β and γ later (and α=1-β-γ, too)

$$\beta(a_x - b_x) + \gamma(a_x - c_x) + t\,d_x = a_x - o_x$$
$$\beta(a_y - b_y) + \gamma(a_y - c_y) + t\,d_y = a_y - o_y$$
$$\beta(a_z - b_z) + \gamma(a_z - c_z) + t\,d_z = a_z - o_z$$

triangle is hit if $\beta \geq 0, \gamma \geq 0, \beta + \gamma \leq 1$

Cramer's Rule

Equations
with unknowns
x, y, z

$$a\,x + b\,y + c\,z = d$$

$$e\,x + f\,y + g\,z = h$$

$$i\,x + j\,y + k\,z = l$$

Cramer's Rule

Equations with unknowns x, y, z

$$a\,x+b\,y+c\,z=d$$
$$e\,x+f\,y+g\,z=h$$
$$i\,x+j\,y+k\,z=l$$

Solution

if $D\neq 0$

$$x=\frac{d(f\,k-g\,j)+b(g\,l-h\,k)+c(h\,j-f\,l)}{D}$$
$$y=\frac{a(h\,k-g\,l)+d(g\,i-e\,k)+c(e\,l-h\,i)}{D}$$
$$z=\frac{a(f\,l-h\,j)+b(h\,i-e\,l)+d(e\,j-f\,i)}{D}$$
$$D=a(f\,k-g\,j)+b(g\,i-e\,k)+c(e\,j-f\,i)$$

IT UNIVERSITY OF COPENHAGEN

Questions

IT UNIVERSITY OF COPENHAGEN

PLY Files

PLY Files

```
ply
format ascii 1.0
comment this is a bunny!!
element vertex 35947
property float32 x
property float32 y
property float32 z
element face 69451
property list uint8 int32 vertex_indices
end_header
-0.0378297 0.12794 0.00447467
-0.0447794 0.128887 0.00190497
-0.0680095 0.151244 0.0371953
…
3 17541 17542 17454
3 17539 17634 17540
3 19612 19503 19502
…
```

PLY Files

```
ply
format ascii 1.0
comment this is a bunny!!
element vertex 35947
property float32 x
property float32 y
property float32 z
element face 69451
property list uint8 int32 vertex_indices
end_header
-0.0378297 0.12794 0.00447467
-0.0447794 0.128887 0.00190497
-0.0680095 0.151244 0.0371953
…
3 17541 17542 17454
3 17539 17634 17540
3 19612 19503 19502
…
```

PLY Files

```
ply
format ascii 1.0
comment this is a bunny!!
element vertex 35947
property float32 x
property float32 y
property float32 z
element face 69451
property list uint8 int32 vertex_indices
end_header
-0.0378297 0.12794 0.00447467
-0.0447794 0.128887 0.00190497
-0.0680095 0.151244 0.0371953

3 17541 17542 17454
3 17539 17634 17540
3 19612 19503 19502
…
```

PLY Files

```
ply
format ascii 1.0
comment this is a bunny!!
element vertex 35947
property float32 x
property float32 y
property float32 z
element face 69451
property list uint8 int32 vertex_indices
end_header
-0.0378297 0.12794 0.00447467
-0.0447794 0.128887 0.00190497
-0.0680095 0.151244 0.0371953

3 17541 17542 17454
3 17539 17634 17540
3 19612 19503 19502
…
```

PLY Files

```
ply
format ascii 1.0
comment this is a bunny!!
element vertex 35947
property float32 x
property float32 y
property float32 z
element face 69451
property list uint8 int32 vertex_indices
end_header
-0.0378297 0.12794 0.00447467
-0.0447794 0.128887 0.00190497
-0.0680095 0.151244 0.0371953

3 17541 17542 17454
3 17539 17634 17540
3 19612 19503 19502
…
```

list length

list elements

PLY Files

```
ply
format ascii 1.0
comment this is a bunny!!
element vertex 35947
property float32 x
property float32 y
property float32 z
element face 69451
property list uint8 int32 vertex_indices
end_header
-0.0378297 0.12794 0.00447467
-0.0447794 0.128887 0.00190497
-0.0680095 0.151244 0.0371953

3 17541 17542 17454
3 17539 17634 17540
3 19612 19503 19502
…
```

list length

list elements

other types:

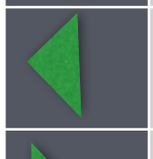
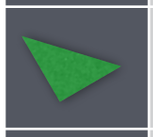
- char (int8)
- uchar (uint8)
- short (int16)
- ushort (uint16)
- int (int32)
- uint (uint32)
- float (float32)
- double (float64)

Data Structure

Vertices

	x	y	z
0	1.23	0.2	12.2
1	3.44	9.47	3.09
2	2.42	3.0	3.0
3	…	…	…
4	…	…	…
5	…	…	…
6	…	…	…
7	…	…	…
8	…	…	…

Triangles

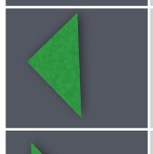
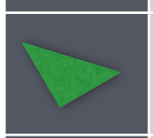
	P_1	P_2	P_3
	3	2	0
	1	4	3
	5	1	7
	7	5	2
	8	1	2
	7	8	6

Data Structure

Vertices

	x	y	z	normal	…
0	1.23	0.2	12.2		
1	3.44	9.47	3.09		
2	2.42	3.0	3.0		
3	…	…	…		
4	…	…	…		
5	…	…	…		
6	…	…	…		
7	…	…	…		
8	…	…	…		

Triangles

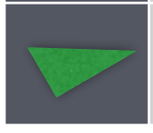
	P_1	P_2	P_3
	3	2	0
	1	4	3
	5	1	7
	7	5	2
	8	1	2
	7	8	6

Data Structure

Vertices

	x	y	z	normal	…
0	1.23	0.2	12.2		
1	3.44	9.47	3.09		
2	2.42	3.0	3.0		
3	…	…	…		
4	…	…	…		
5	…	…	…		
6	…	…	…		
7	…	…	…		
8	…	…	…		

Triangles

	P_1	P_2	P_3	…
	3	2	0	
	1	4	3	
	5	1	7	
	7	5	2	
	8	1	2	
	7	8	6	

Additional Data

```
ply
format ascii 1.0
comment this is a bunny!!
element vertex 35947
comment has texture coords and normals
property float x
property float y
property float z
property float nx
property float ny
property float nz
property float u
property float v
property float fooness
property float barness
element face 69451
property list uint uint vertex_indices
end_header
…
```

Additional Data

```
ply
format ascii 1.0
comment this is a bunny!!
element vertex 35947
comment has texture coords and normals
property float x
property float y
property float z
property float nx
property float ny
property float nz
property float u
property float v
property float fooness
property float barness
element face 69451
property list uint uint vertex_indices
end_header
…
```

normal vector

Additional Data

```
ply
format ascii 1.0
comment this is a bunny!!
element vertex 35947
comment has texture coords and normals
property float x
property float y
property float z
property float nx
property float ny
property float nz
property float u
property float v
property float fooness
property float barness
element face 69451
property list uint uint vertex_indices
end_header
…
```

normal vector

texture coordinates

Parsing ASCII Files

- FParsec: parser combinator library
- http://www.quanttec.com/fparsec/

- Alternatively: The file format is simple enough to write a parser without a library.

Binary Format

ASCII

```
ply
format ascii 1.0
…
end_header
…
```

Binary Format

ASCII

```
ply
format ascii 1.0
…
end_header
…
```

PLY files may also come in binary form

```
ply
format binary_big_endian 1.0
…
end_header
…
```

```
ply
format binary_little_endian 1.0
…
end_header
…
```

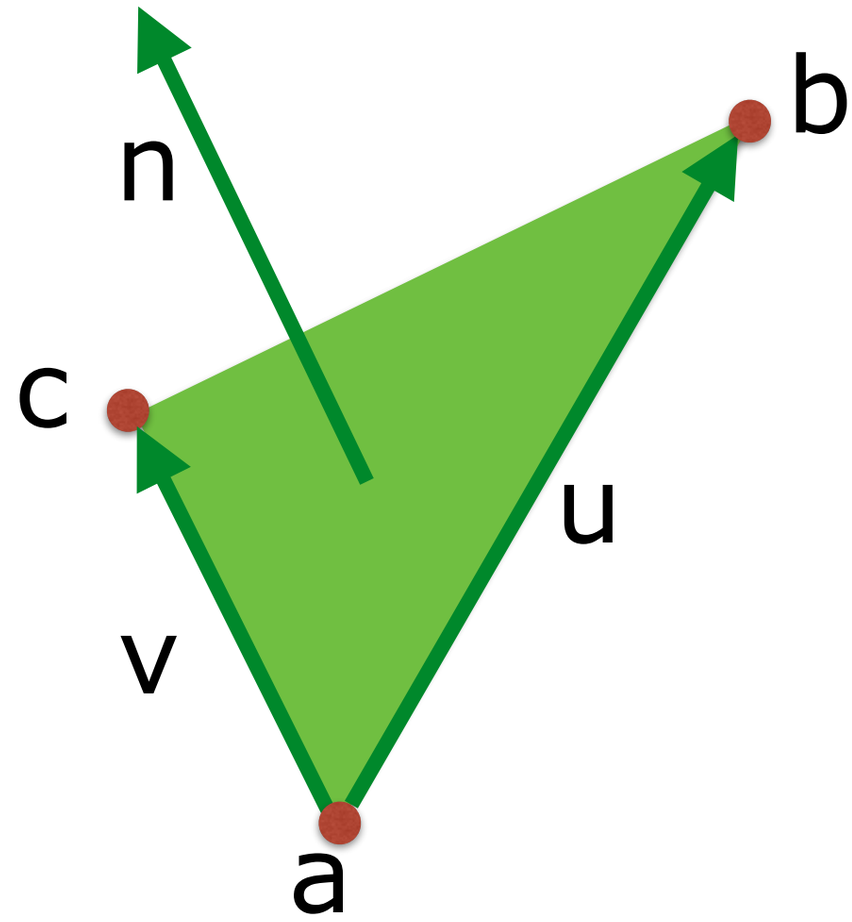
But we will only use the ASCII format

IT UNIVERSITY OF COPENHAGEN

Questions

Smooth Shading

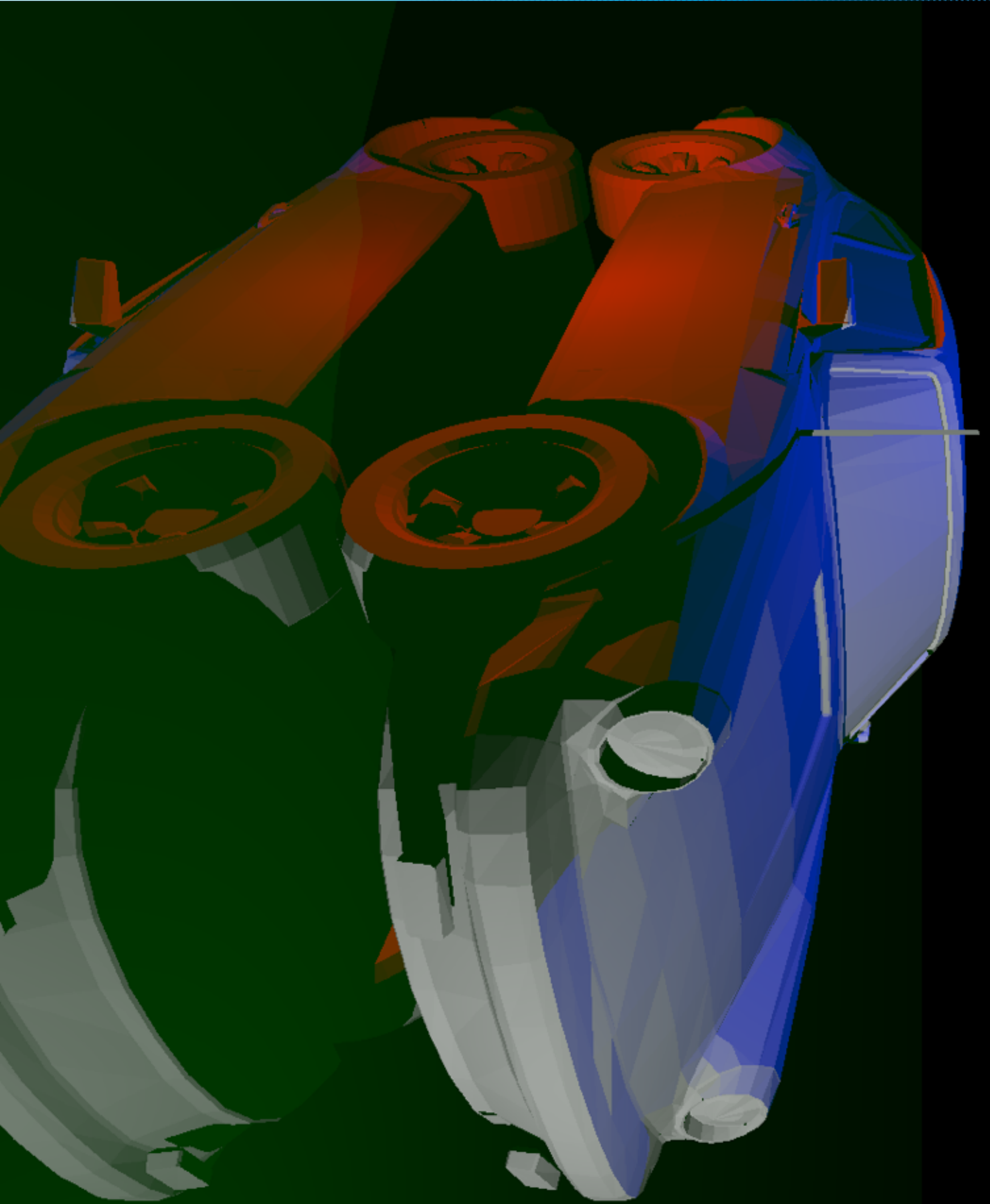
Calculating Normals

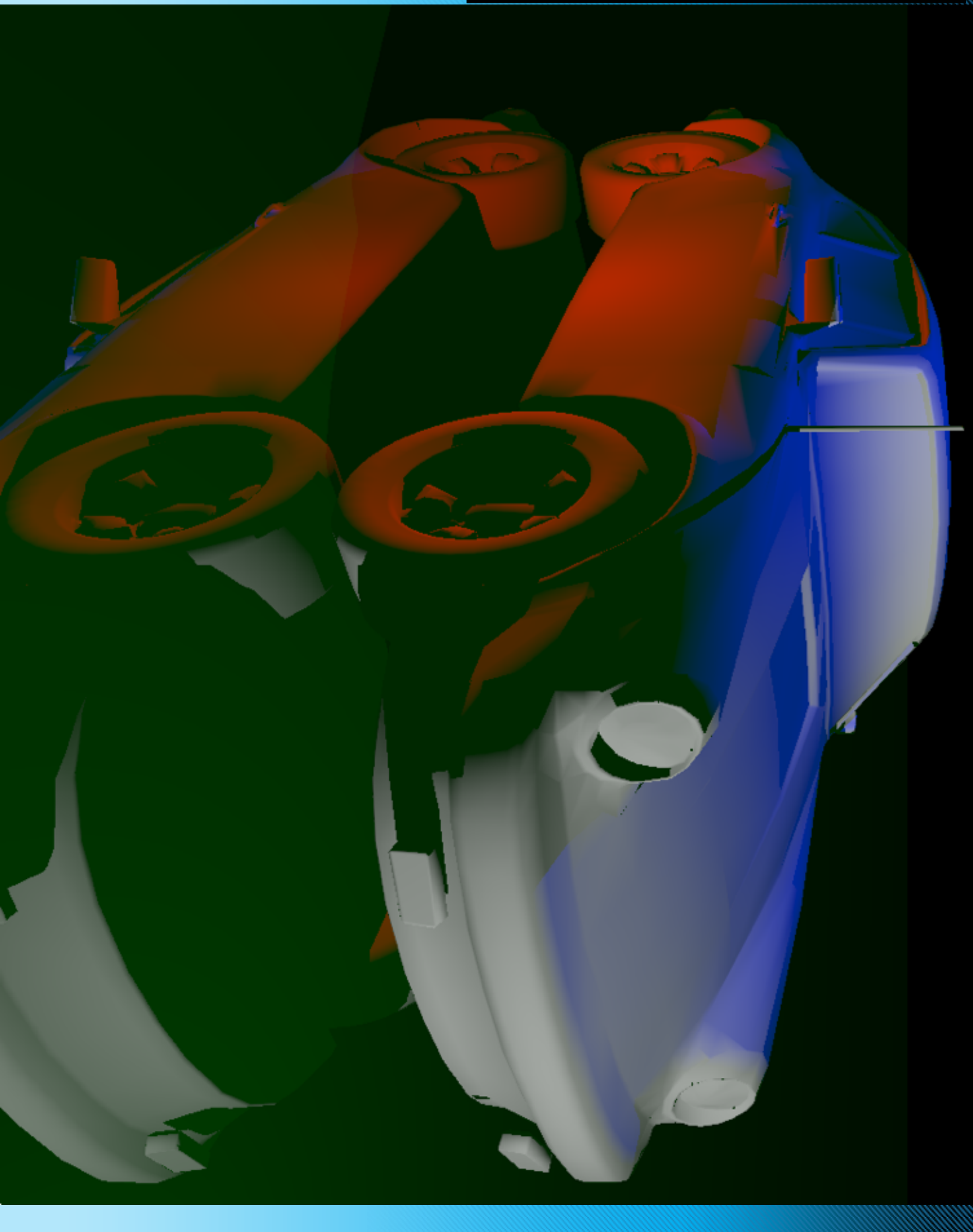


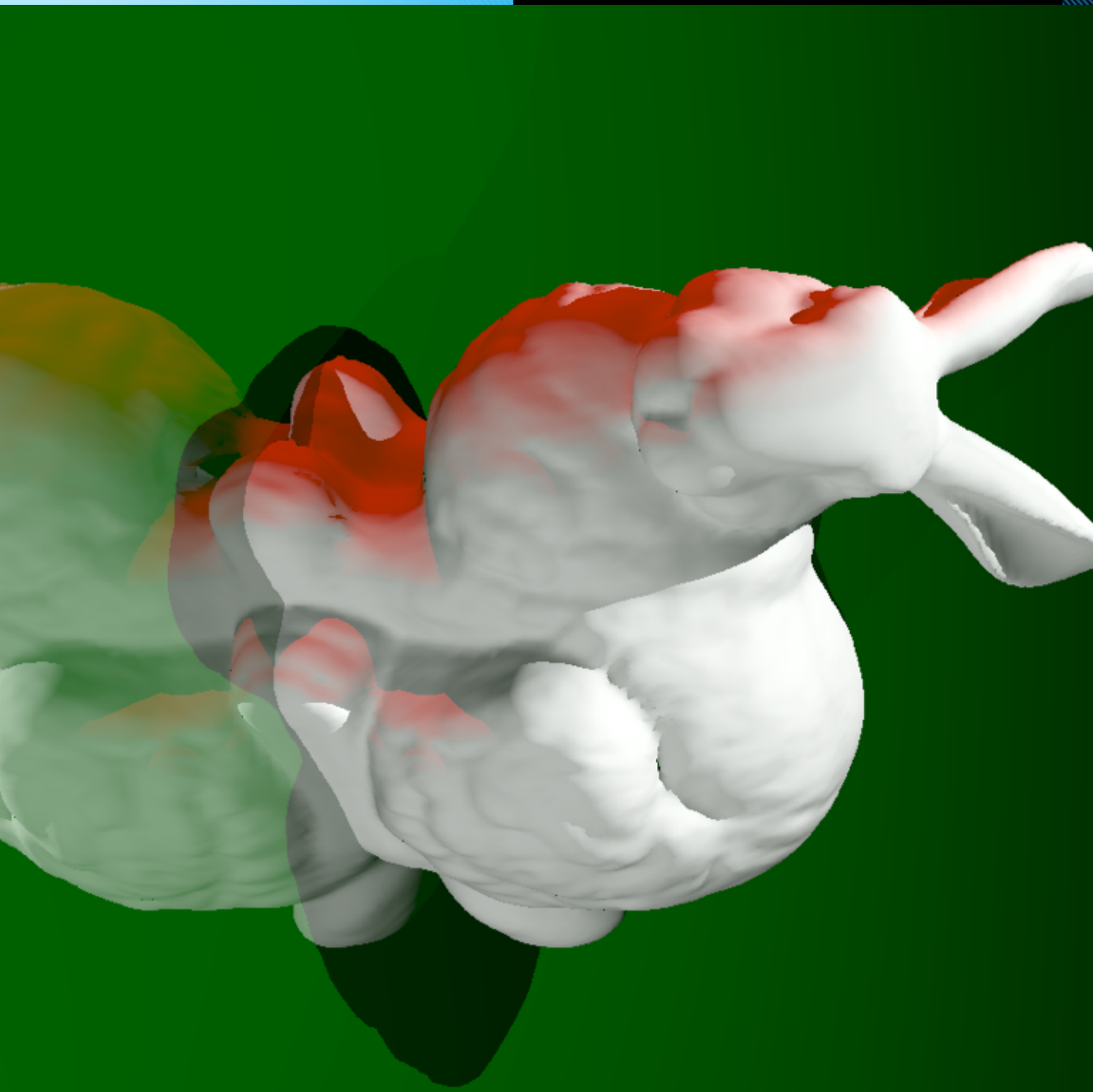
$$u = b - a$$

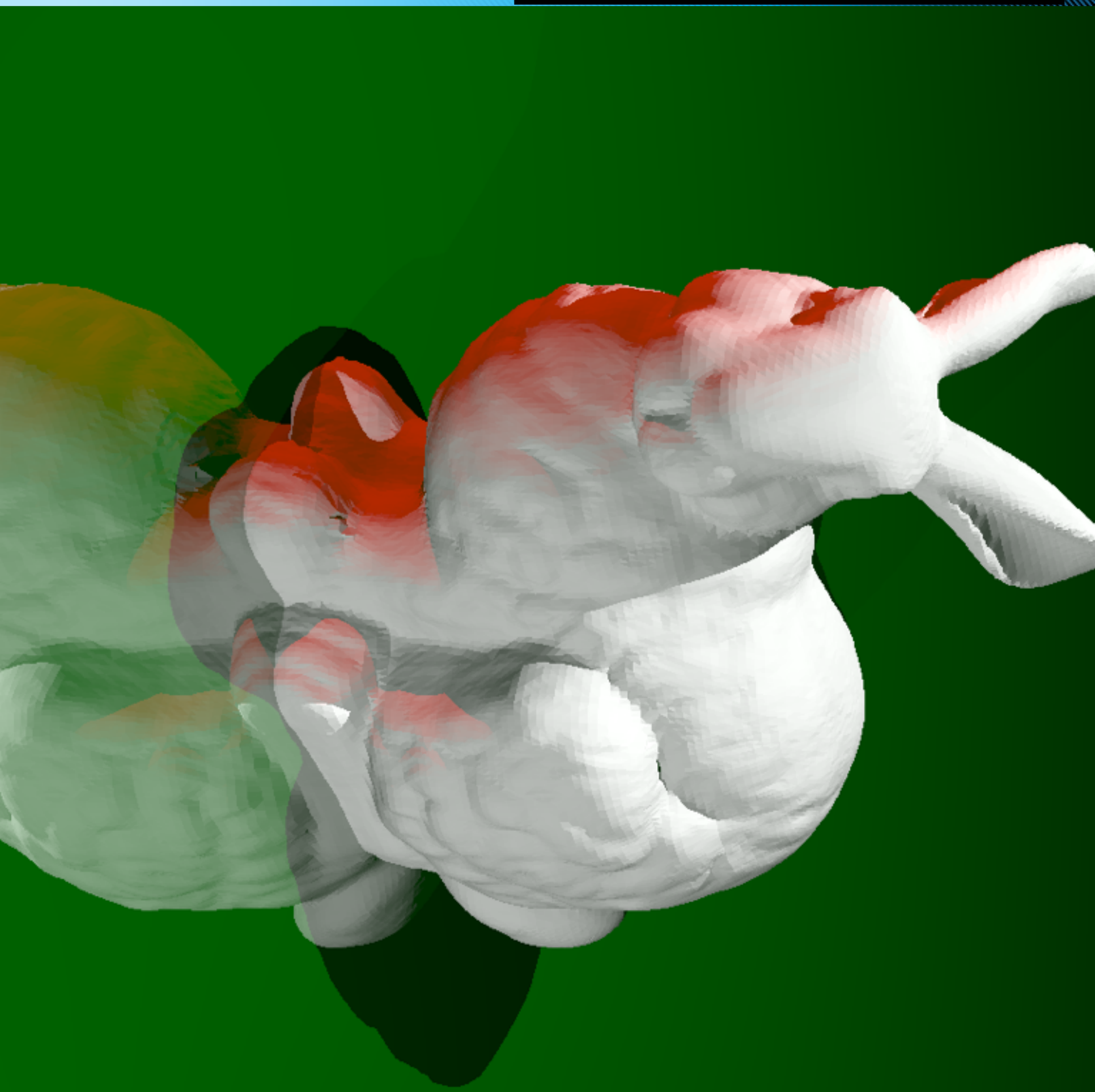
$$v = c - a$$

$$n = \frac{u \times v}{|u \times v|}$$

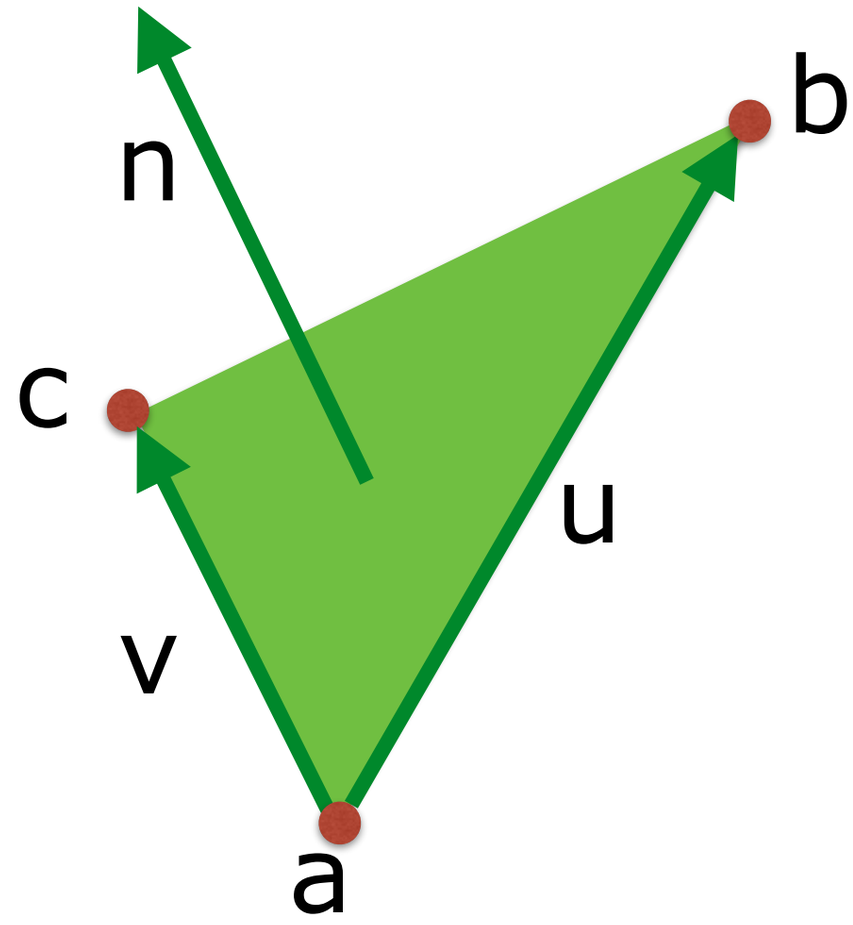






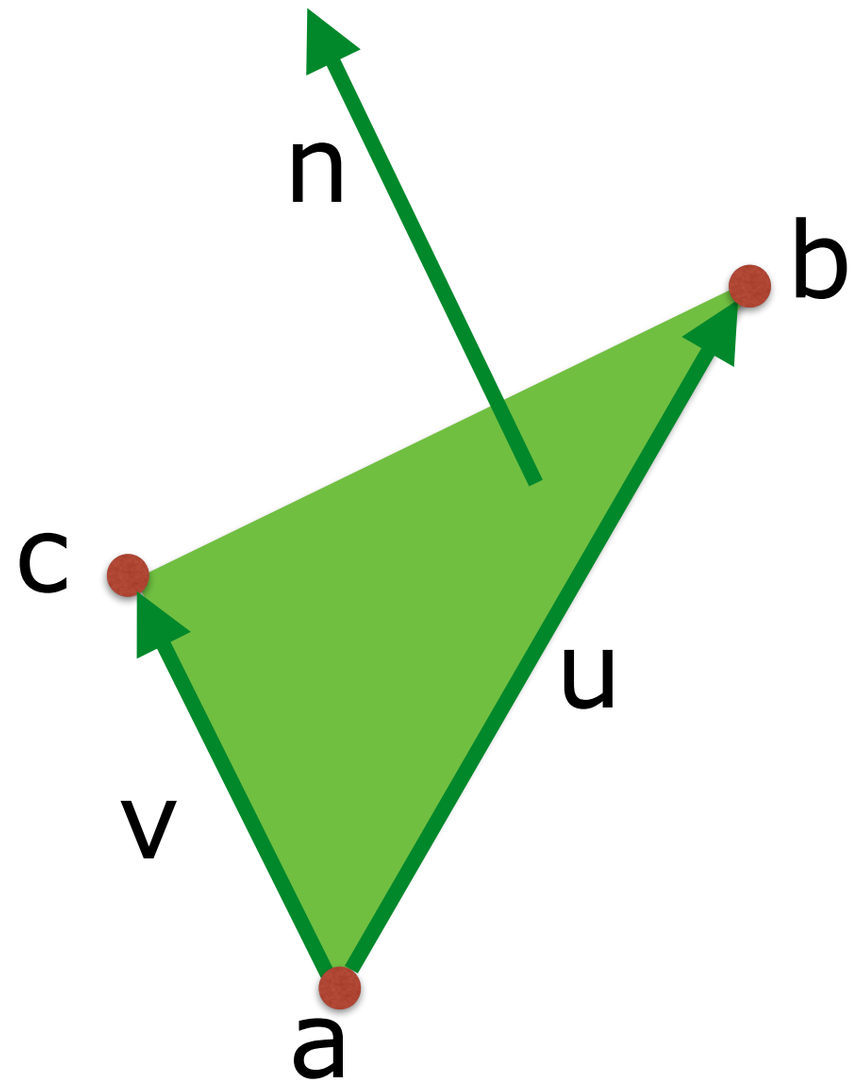


Flat Shading



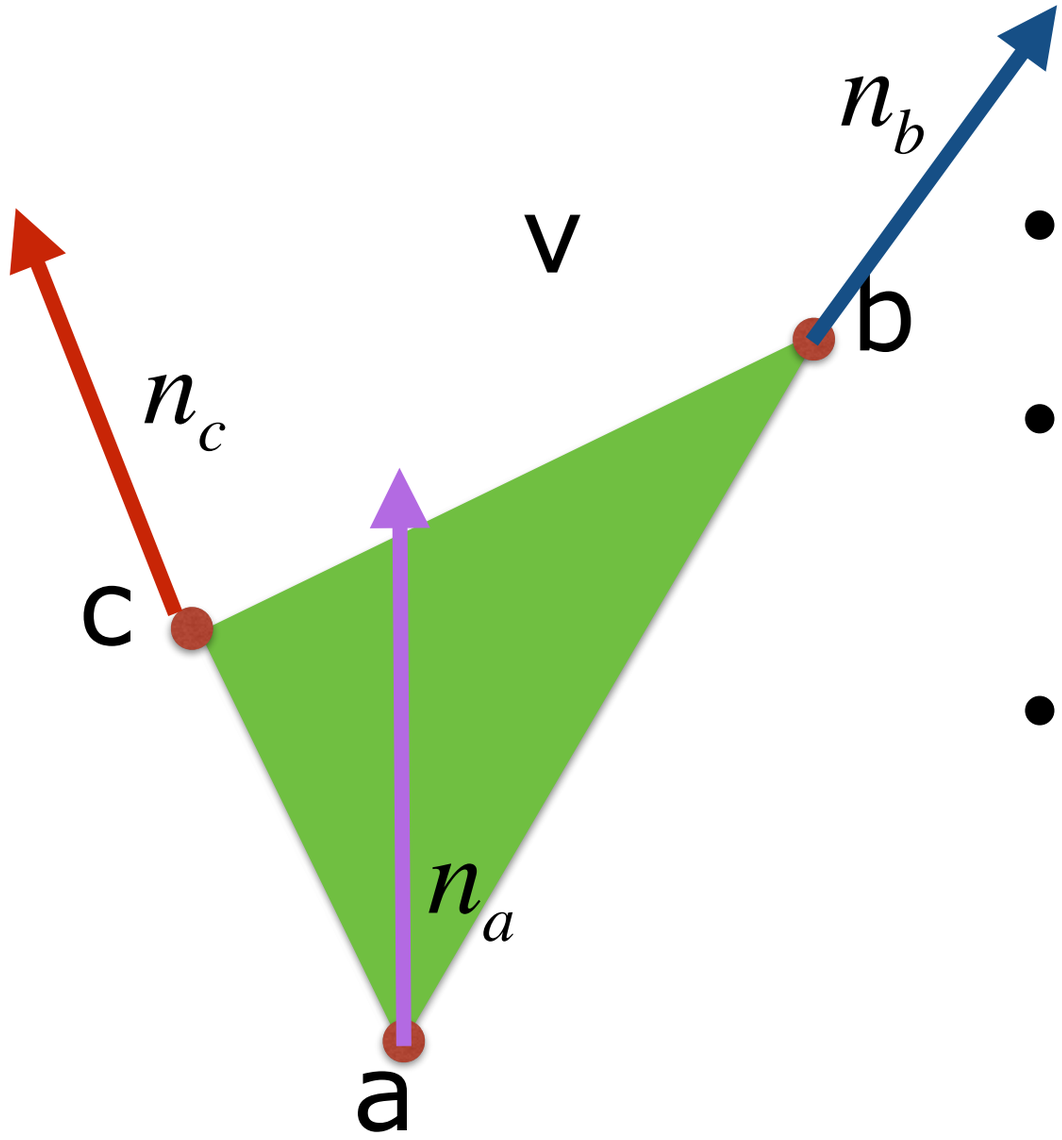
- each triangle has a single normal vector
- every point on a triangle is shaded the same way
- individual triangles are recognisable

Flat Shading



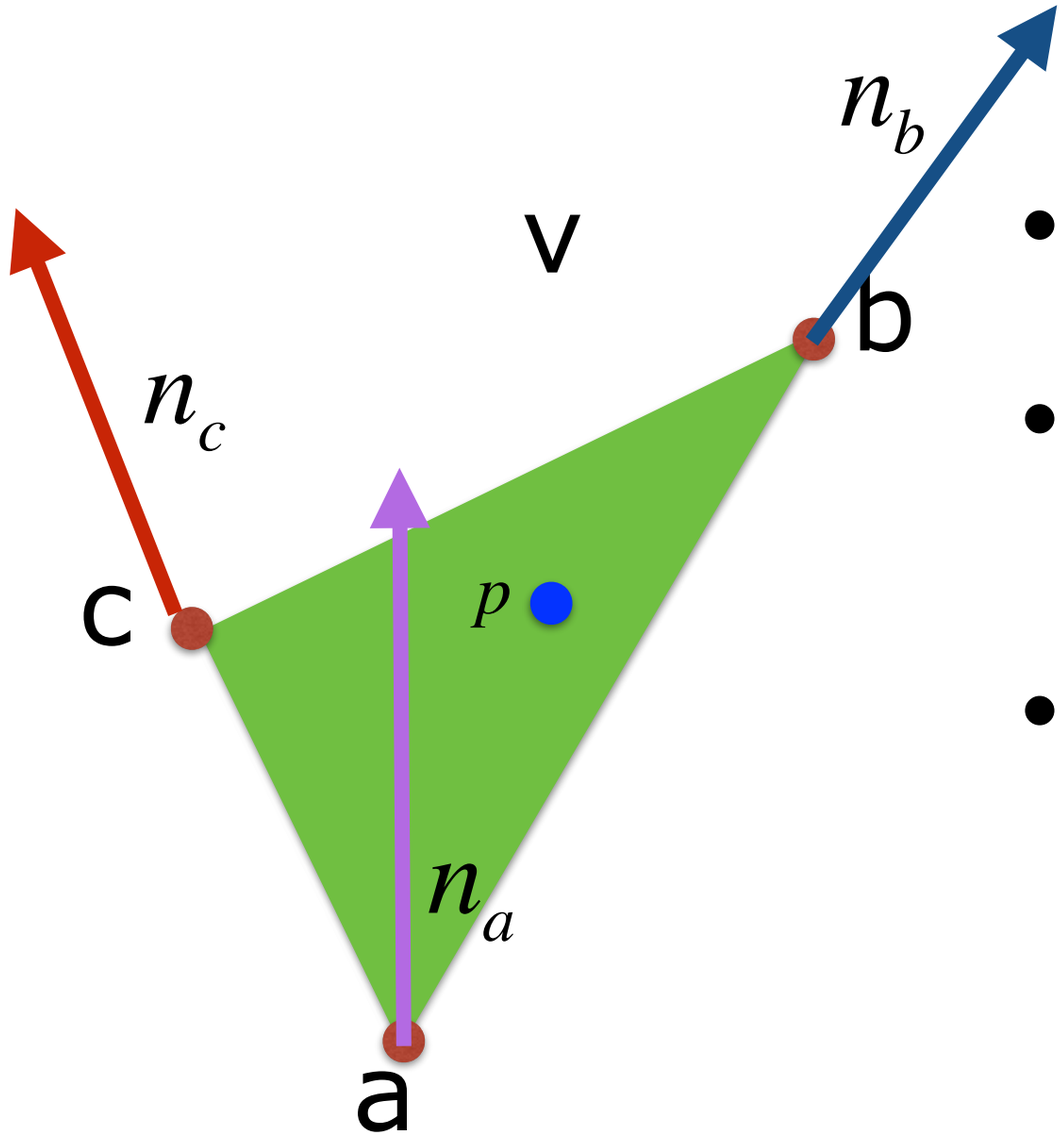
- each triangle has a single normal vector
- every point on a triangle is shaded the same way
- individual triangles are recognisable

Smooth Shading



- each vertex has a normal
- interpolate the normals of the vertices
- the closer the hit point p is to a vertex q, the closer p's normal is to q's normal

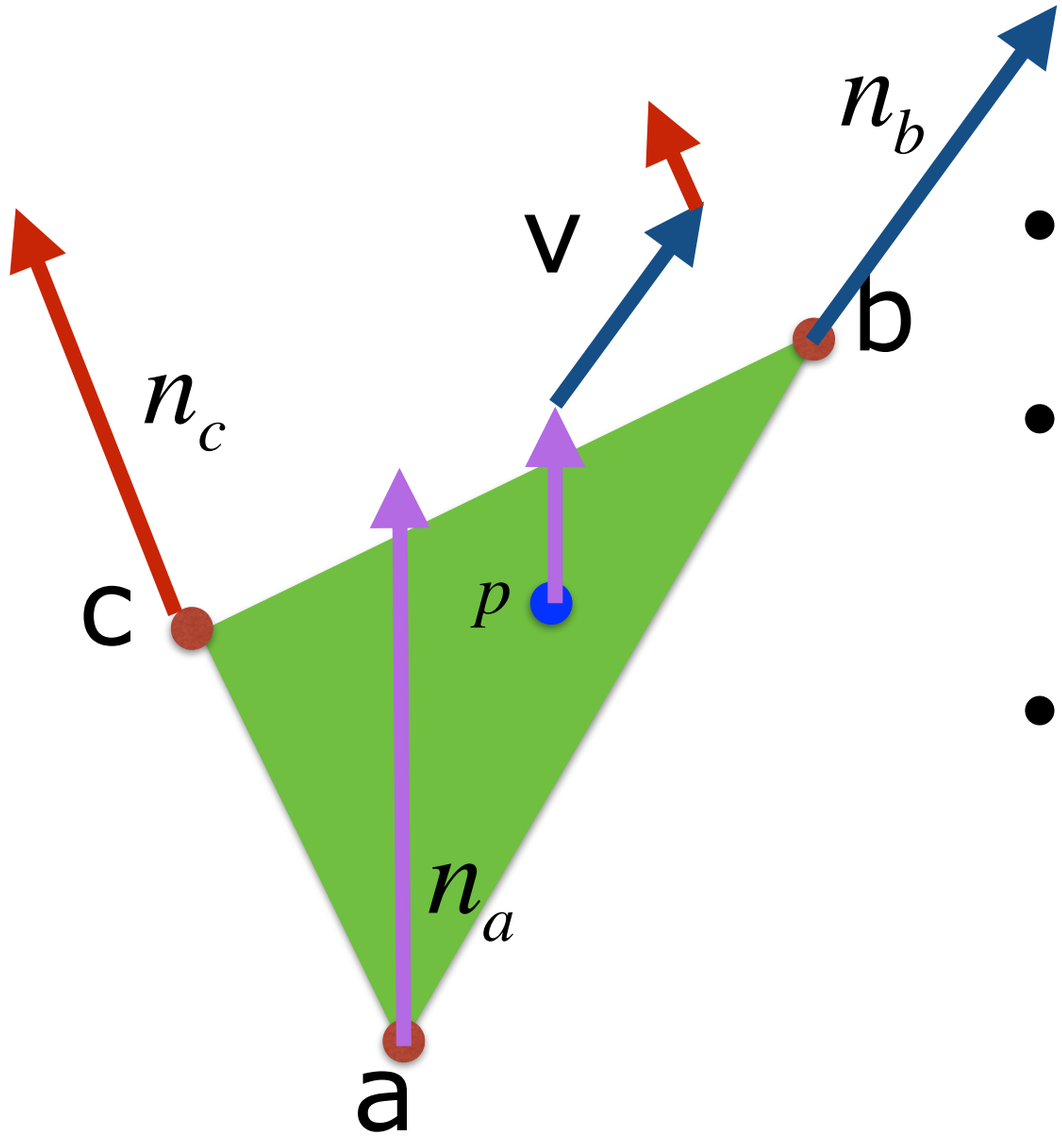
Smooth Shading



$$p = \alpha a + \beta b + \gamma c$$

- each vertex has a normal
- interpolate the normals of the vertices
- the closer the hit point p is to a vertex q, the closer p's normal is to q's normal

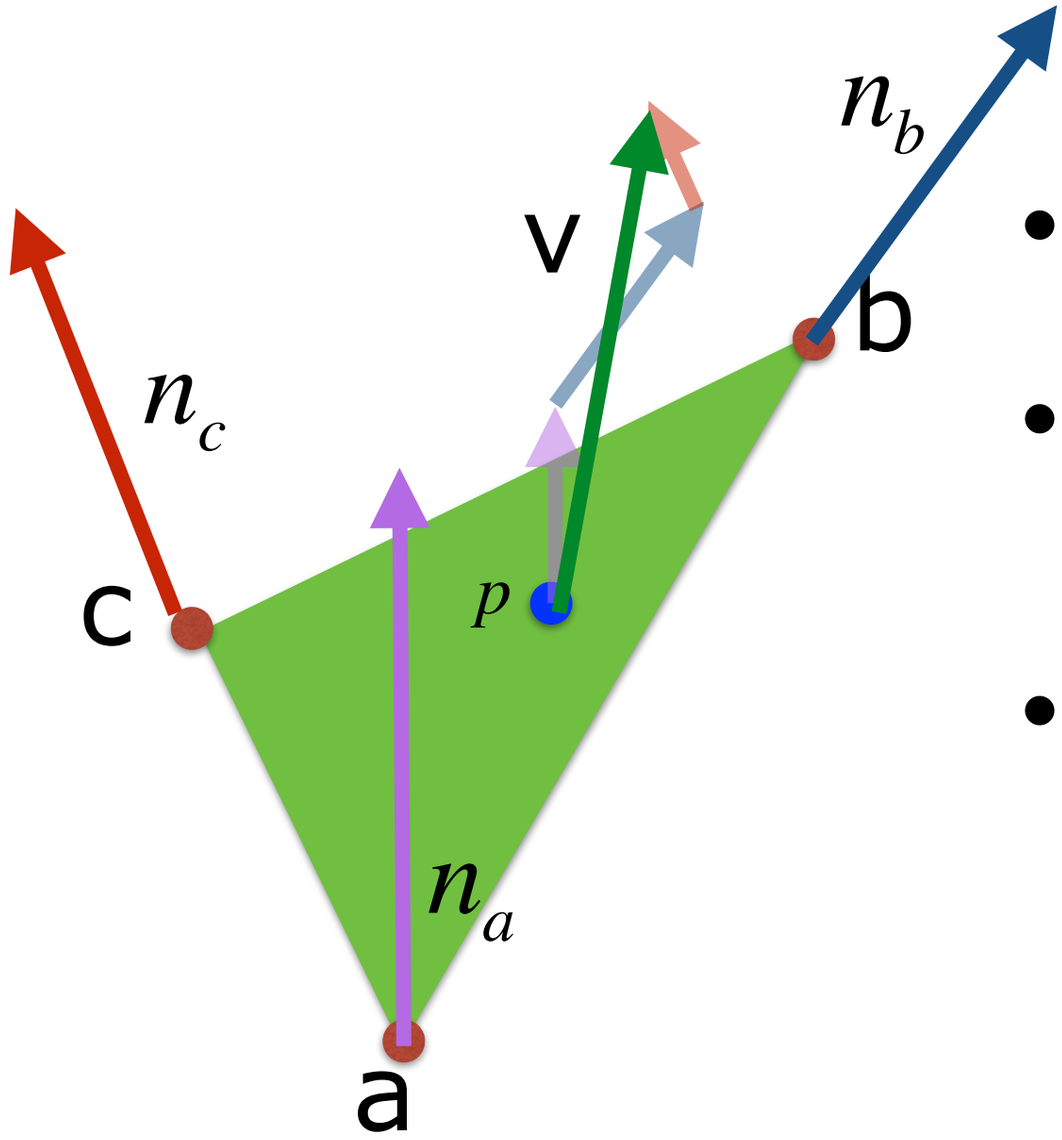
Smooth Shading



$$p = \alpha a + \beta b + \gamma c$$

- each vertex has a normal
- interpolate the normals of the vertices
- the closer the hit point p is to a vertex q, the closer p's normal is to q's normal

Smooth Shading



- each vertex has a normal
- interpolate the normals of the vertices
- the closer the hit point p is to a vertex q, the closer p's normal is to q's normal

$$p = \alpha a + \beta b + \gamma c$$

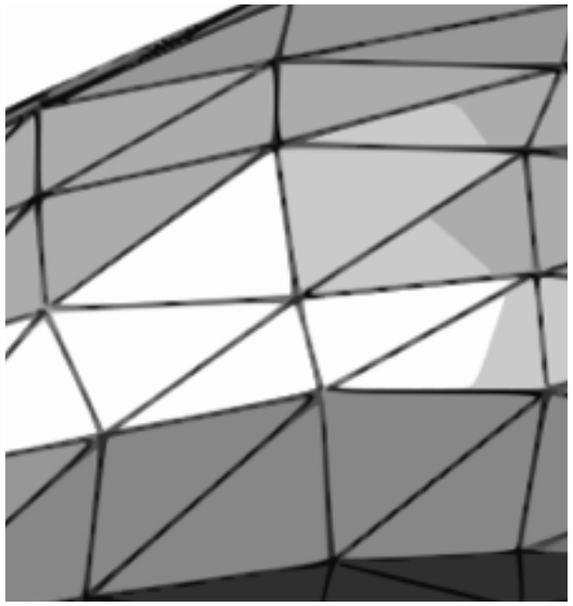
$$v = \alpha n_a + \beta n_b + \gamma n_c$$

v needs to be normalised

$$n = v / |v|$$

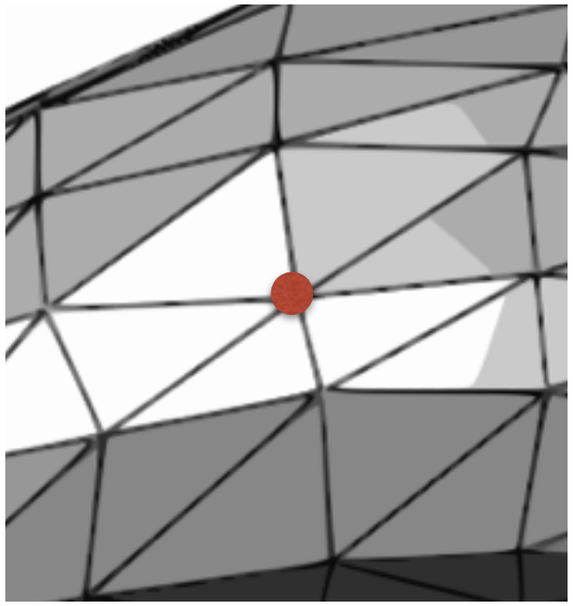
Normals of Vertices

1. may be given in PLY file (nx, ny, nz)
2. can be computed by computing the average of the triangles' normals



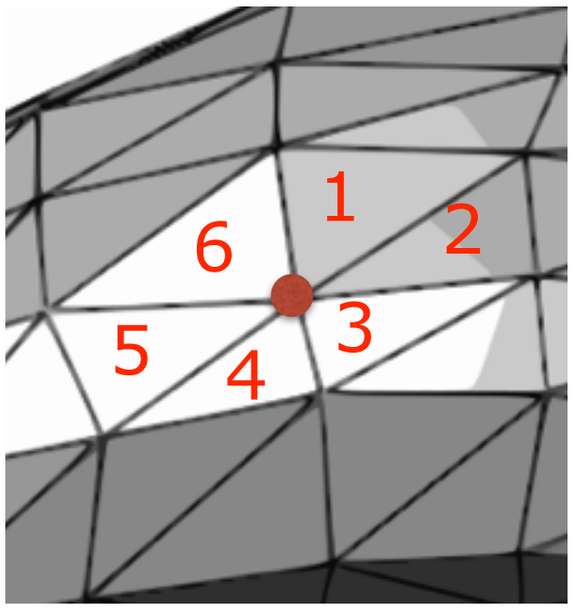
Normals of Vertices

1. may be given in PLY file (nx, ny, nz)
2. can be computed by computing the average of the triangles' normals



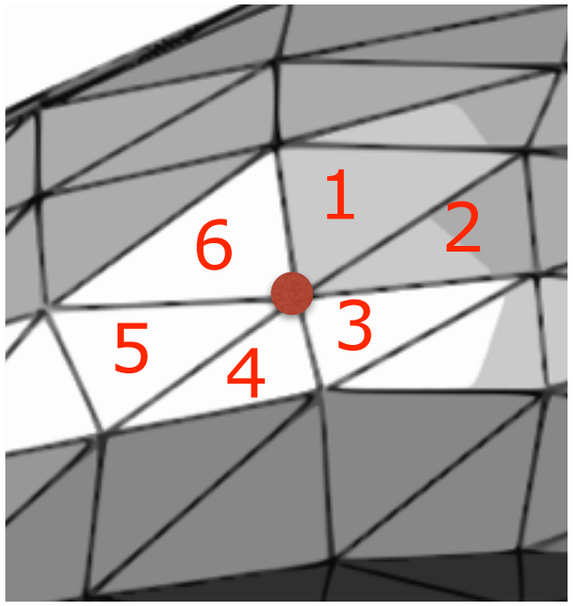
Normals of Vertices

1. may be given in PLY file (nx, ny, nz)
2. can be computed by computing the average of the triangles' normals



Normals of Vertices

1. may be given in PLY file (nx, ny, nz)
2. can be computed by computing the average of the triangles' normals



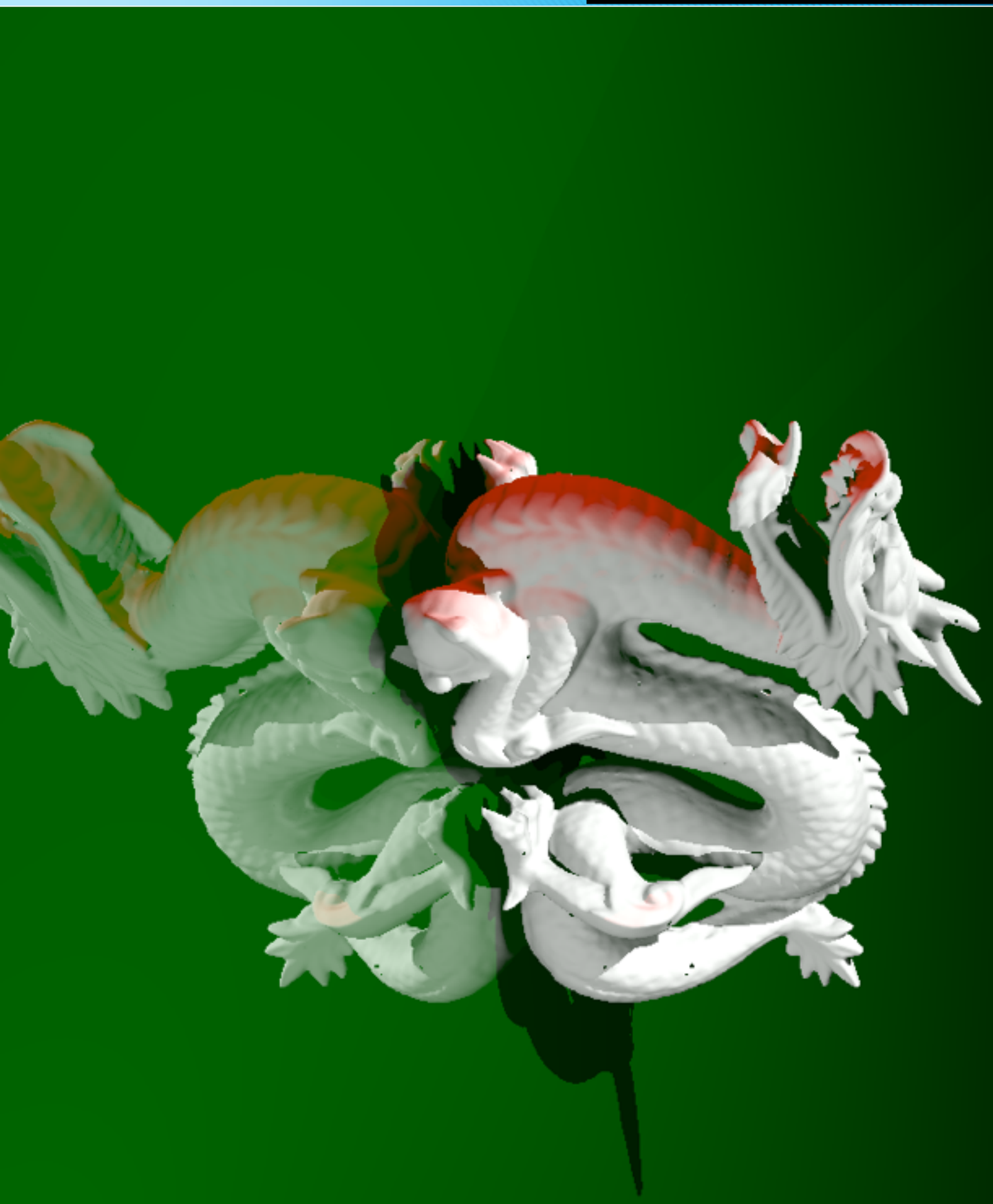
normal of vertex •

$$n = \frac{n_1 + n_2 + n_3 + n_4 + n_5 + n_6}{\left| n_1 + n_2 + n_3 + n_4 + n_5 + n_6 \right|}$$

$n_i =$ normal of triangle i

Smooth shading does not always improve the rendered image!

Next version of API has option to turn smooth shading on/off



IT UNIVERSITY OF COPENHAGEN

Questions

Acceleration Structures

Acceleration Structures

1. Axis Aligned Bounding Boxes

2. kd-Trees

Axis Aligned Bounding Boxes

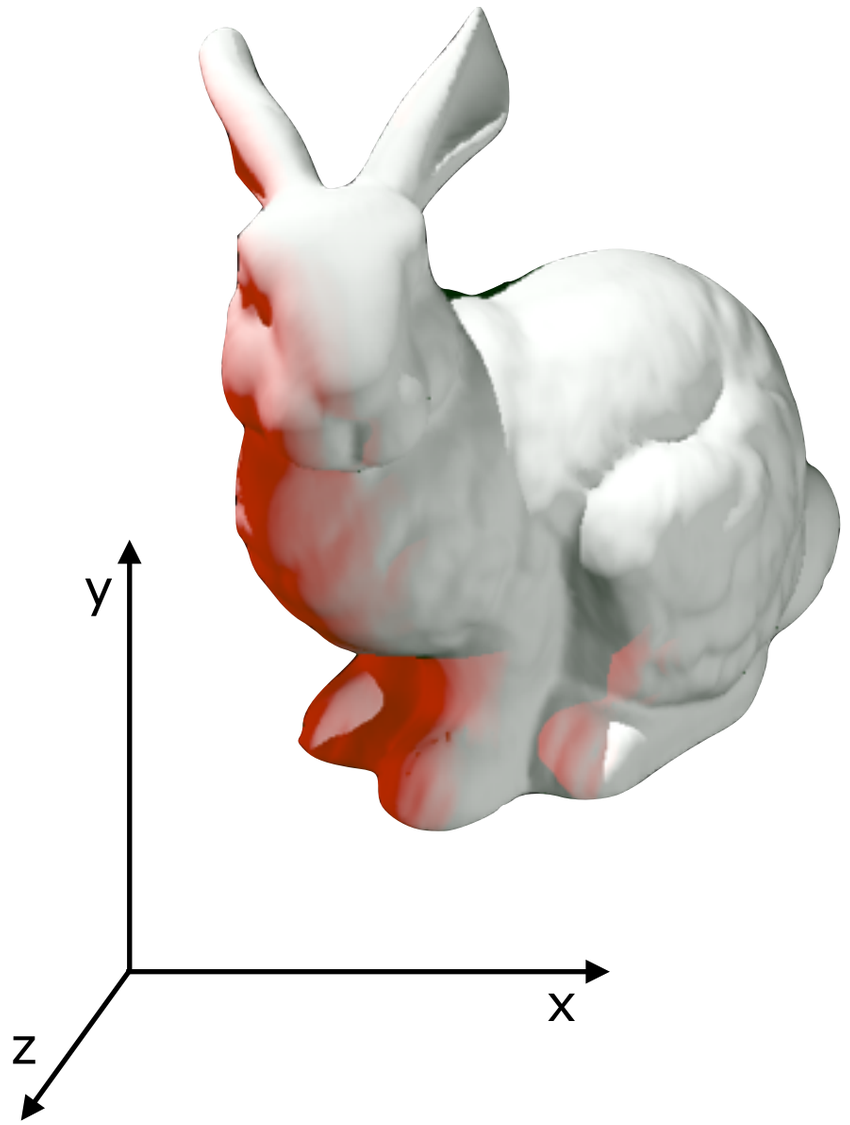
- For shapes with expensive hit function

- Check intersection with bounding box first:

  A. If this check succeeds, continue with normal hit function of the shape.

  B. Otherwise, there will be no hit.

- Also used for constructing kd-trees



Axis Aligned Bounding Boxes

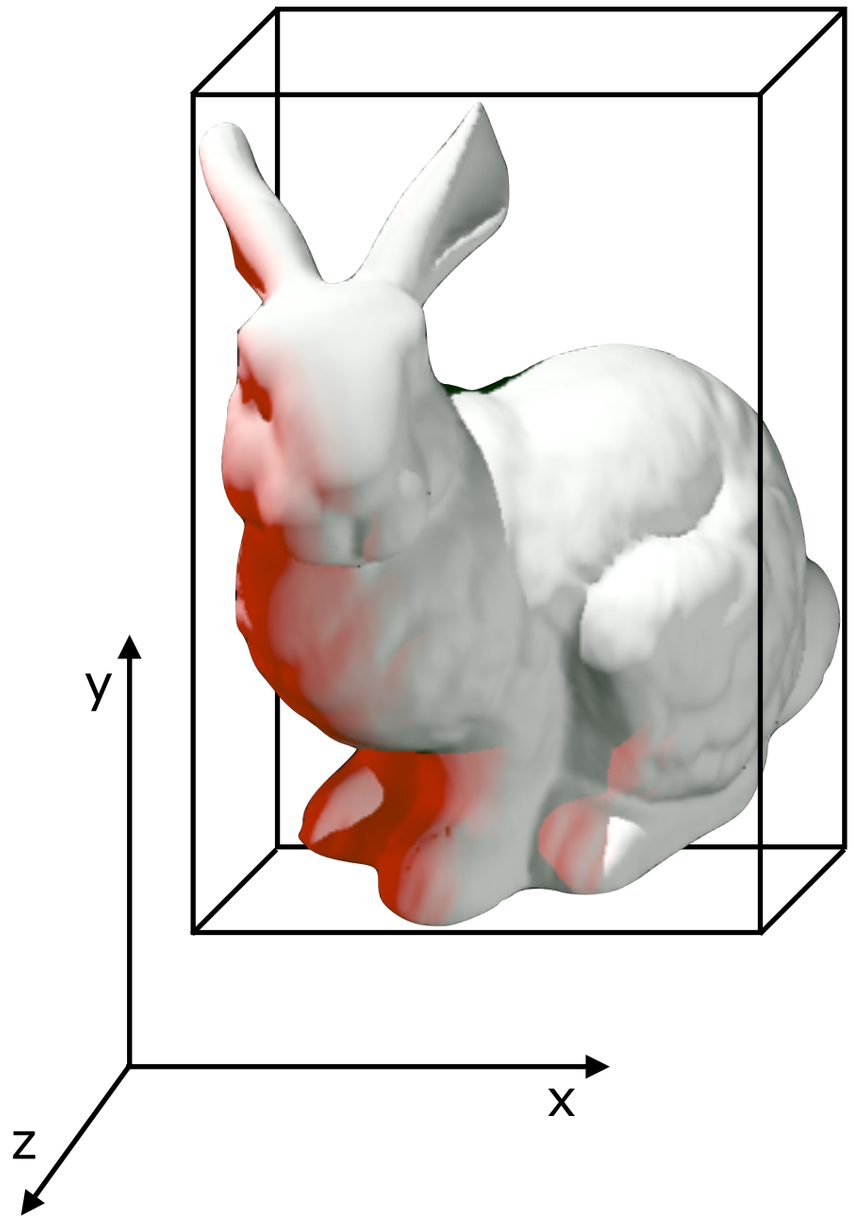
- For shapes with expensive hit function

- Check intersection with bounding box first:

  A. If this check succeeds, continue with normal hit function of the shape.

  B. Otherwise, there will be no hit.

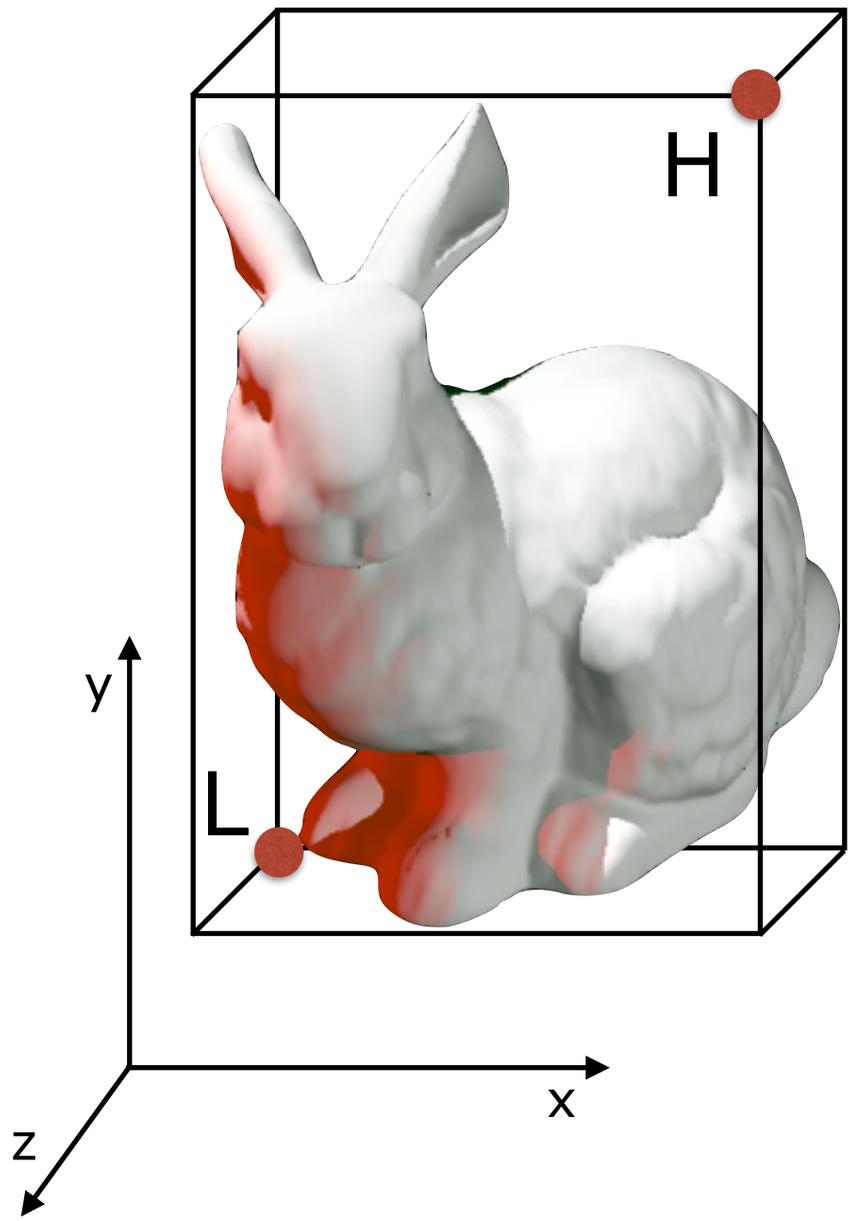
- Also used for constructing kd-trees



Compute Bounding Box

$$L_x = \min\{\text{x coordinates}\} - \varepsilon$$
$$L_y = \min\{y \text{ coordinates}\} - \varepsilon$$
$$L_z = \min\{z \text{ coordinates}\} - \varepsilon$$

$$H_x = \max\{\text{x coordinates}\} + \varepsilon$$
$$H_y = \max\{y \text{ coordinates}\} + \varepsilon$$
$$H_z = \max\{z \text{ coordinates}\} + \varepsilon$$



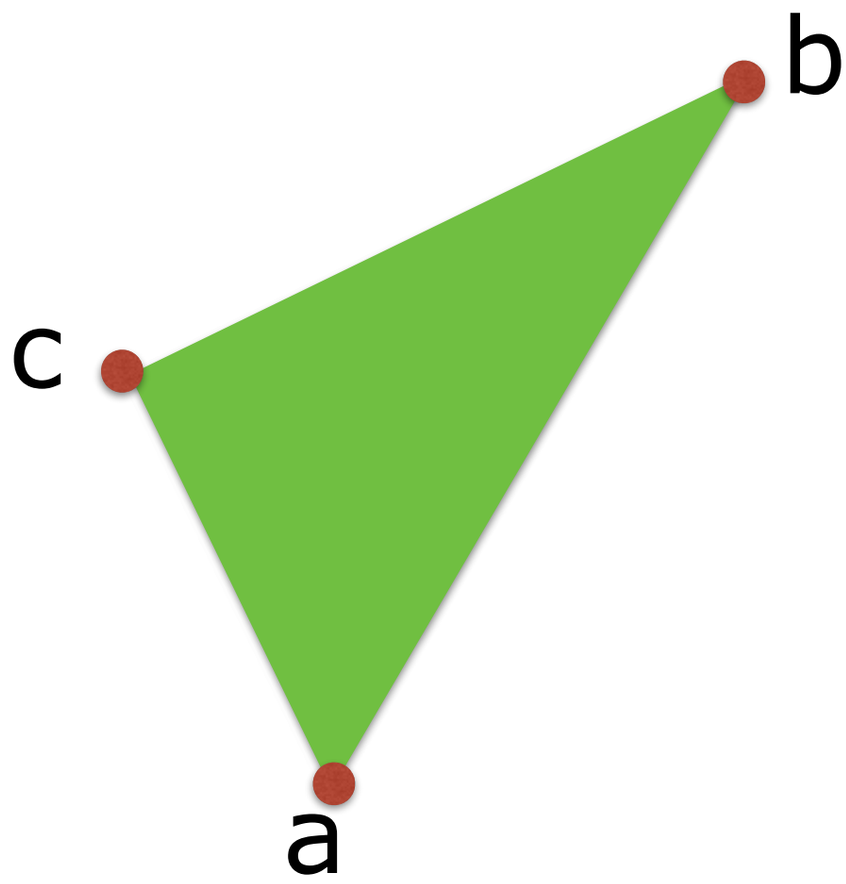
$\varepsilon =$ very small value

→ avoid bounding box with volume 0

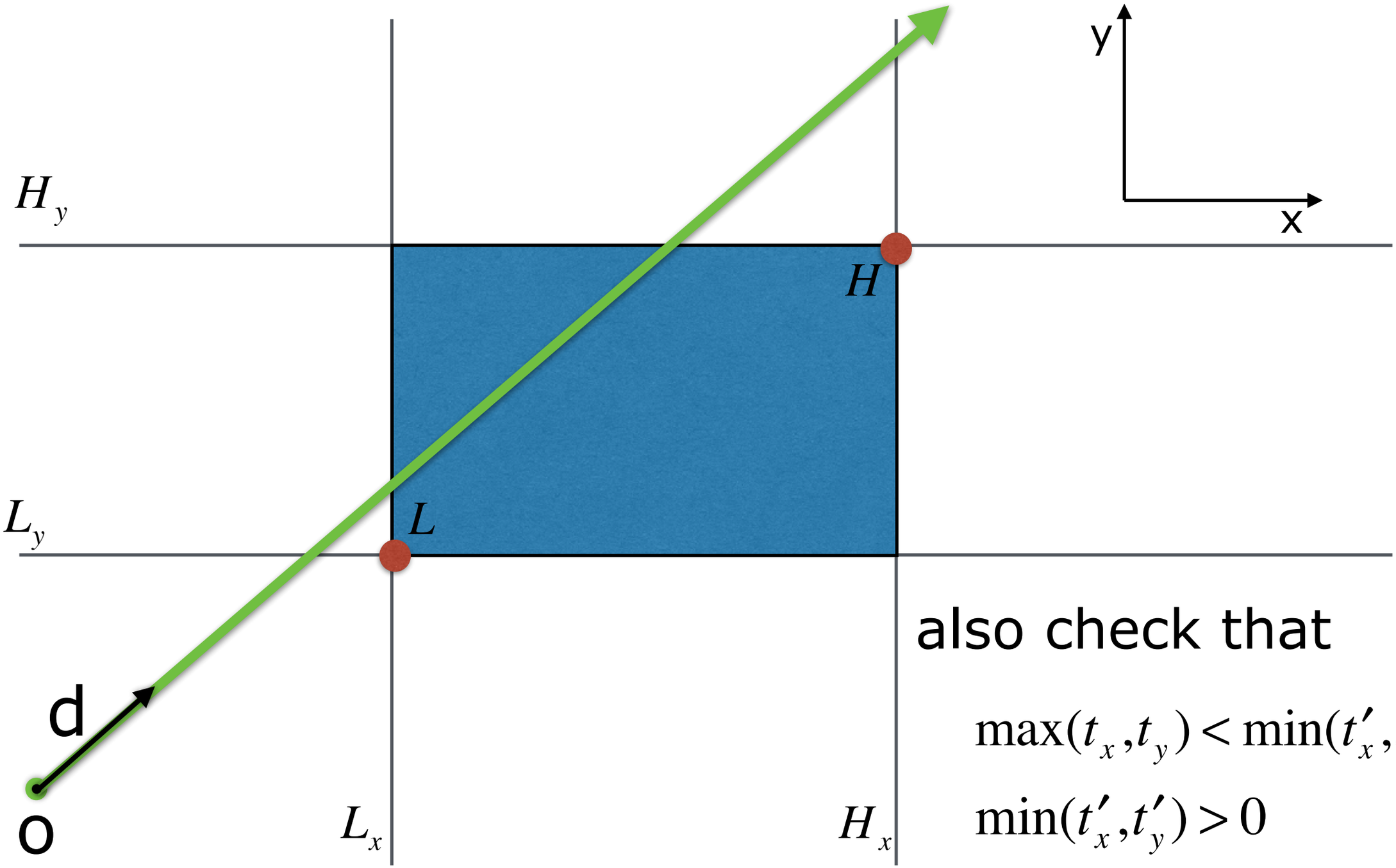
Example: Triangle

$$L_x = \min(a_x, b_x, c_x) - \varepsilon$$
$$L_y = \min(a_y, b_y, c_y) - \varepsilon$$
$$L_z = \min(a_z, b_z, c_z) - \varepsilon$$

$$H_x = \max(a_x, b_x, c_x) + \varepsilon$$
$$H_y = \max(a_y, b_y, c_y) + \varepsilon$$
$$H_z = \max(a_z, b_z, c_z) + \varepsilon$$



Ray Intersection (2D)

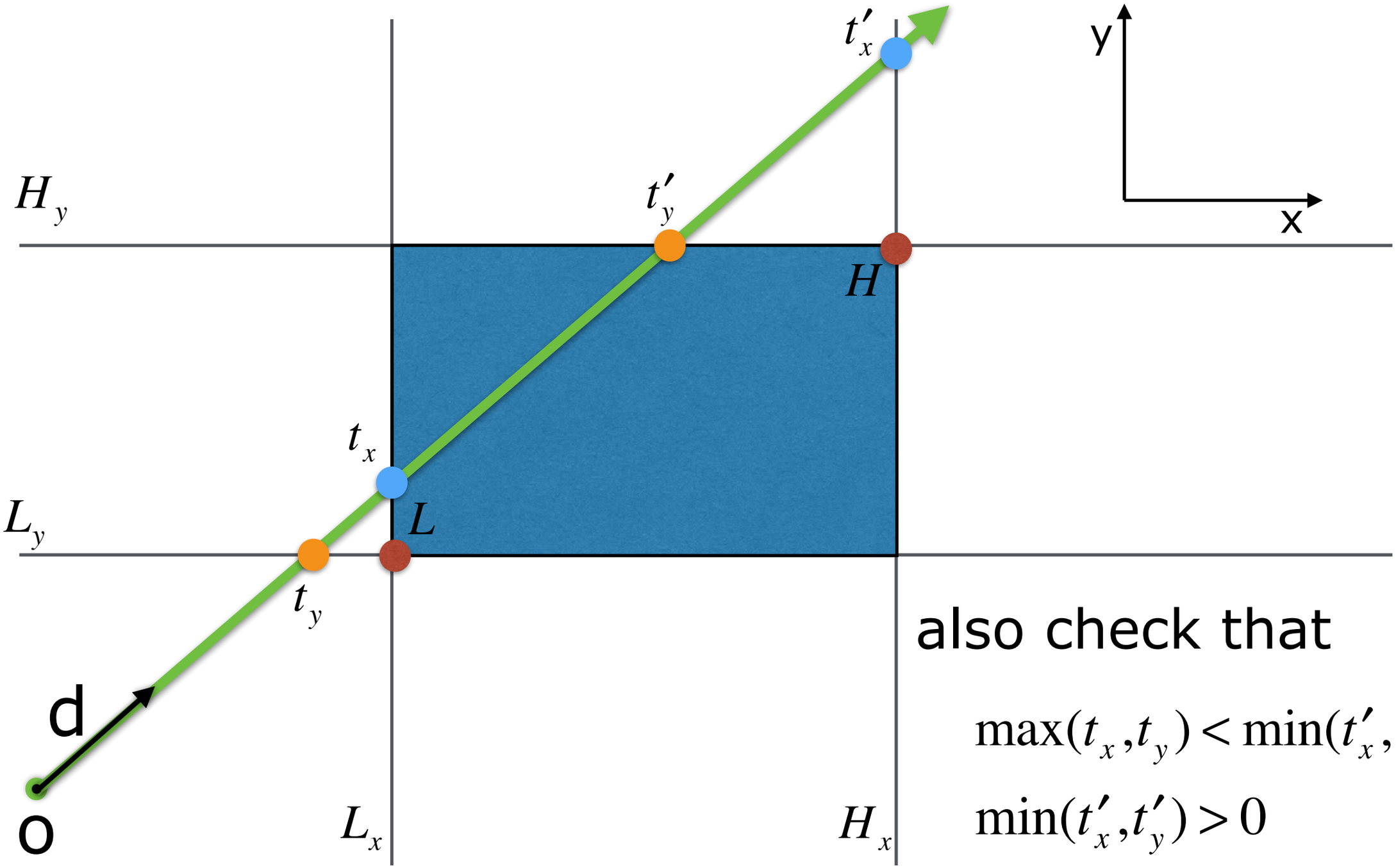


also check that

$$\max(t_x, t_y) < \min(t'_x, t'_y)$$

$$\min(t'_x, t'_y) > 0$$

Ray Intersection (2D)

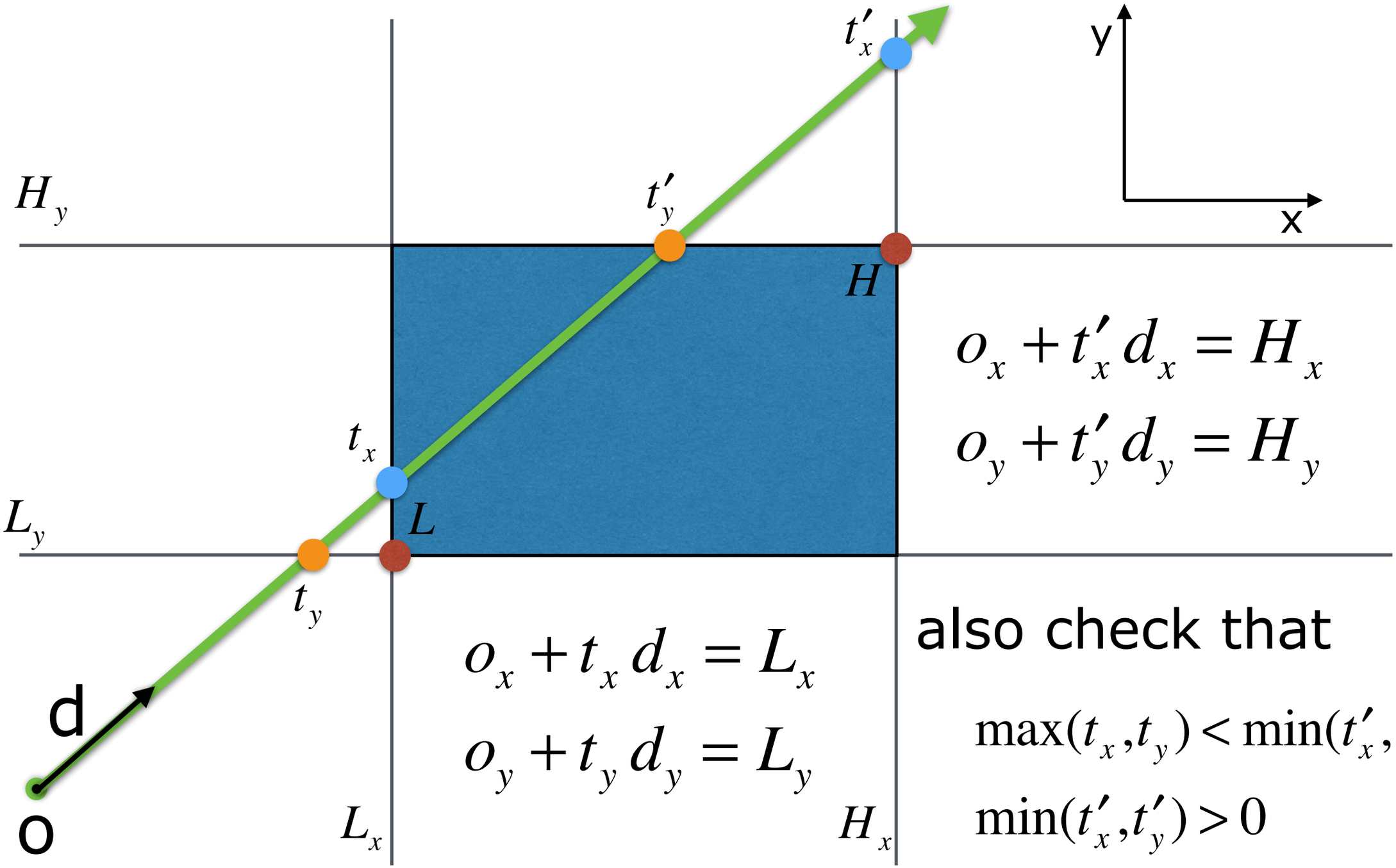


also check that

$$\max(t_x, t_y) < \min(t'_x, t'_y)$$

$$\min(t'_x, t'_y) > 0$$

Ray Intersection (2D)



$$o_x + t'_x d_x = H_x$$

$$o_y + t'_y d_y = H_y$$

$$o_x + t_x d_x = L_x$$

$$o_y + t_y d_y = L_y$$

also check that

$$\max(t_x, t_y) < \min(t'_x, t'_y)$$

$$\min(t'_x, t'_y) > 0$$

Ray Intersection

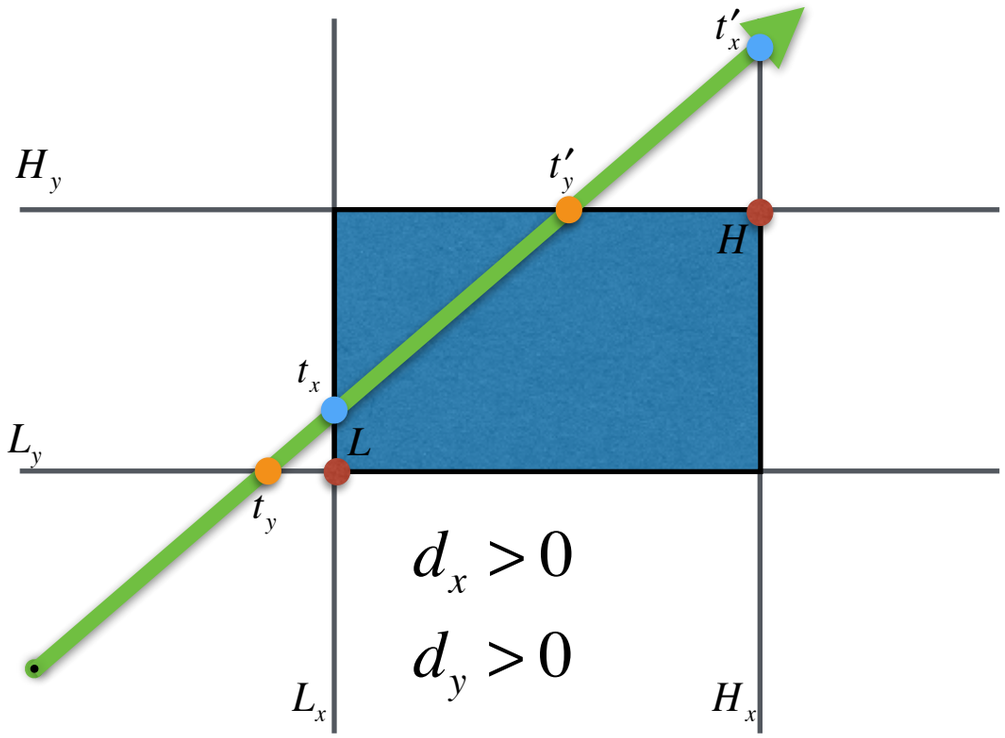
- this calculation only works if $d_x, d_y > 0$
- if $d_x < 0$ then

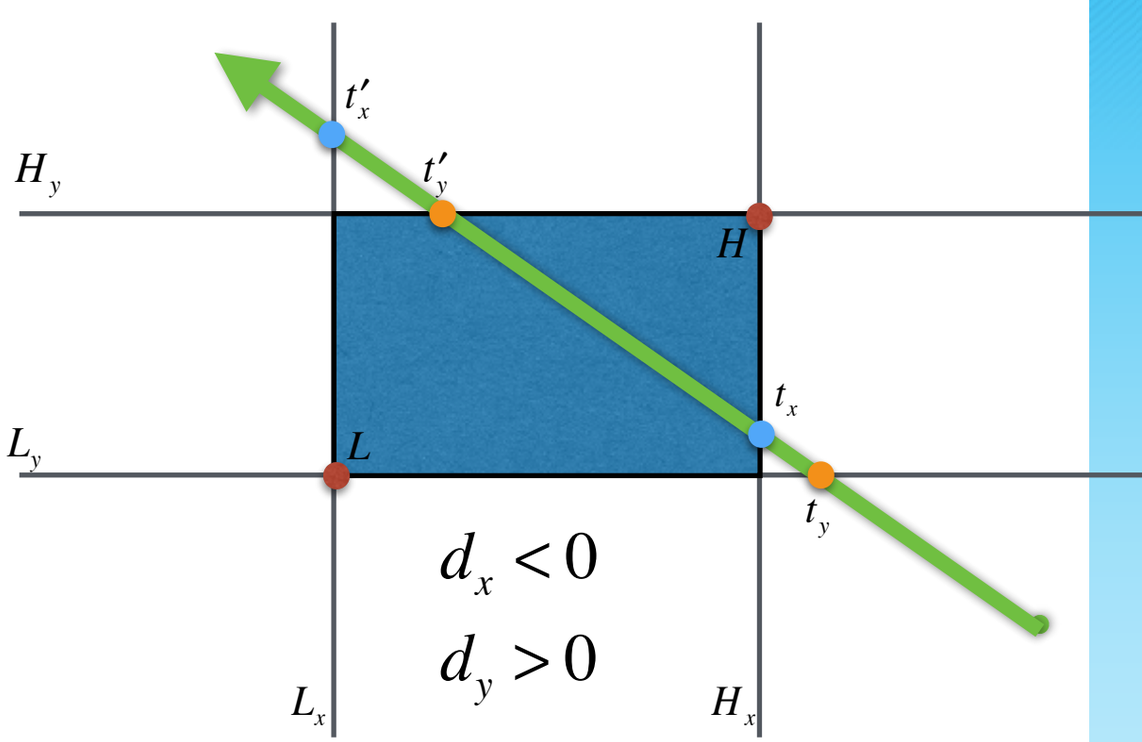
$$o_x + t_x d_x = H_x \qquad o_x + t_x d_x = L_x$$

instead of

$$o_x + t'_x d_x = L_x \qquad o_x + t'_x d_x = H_x$$

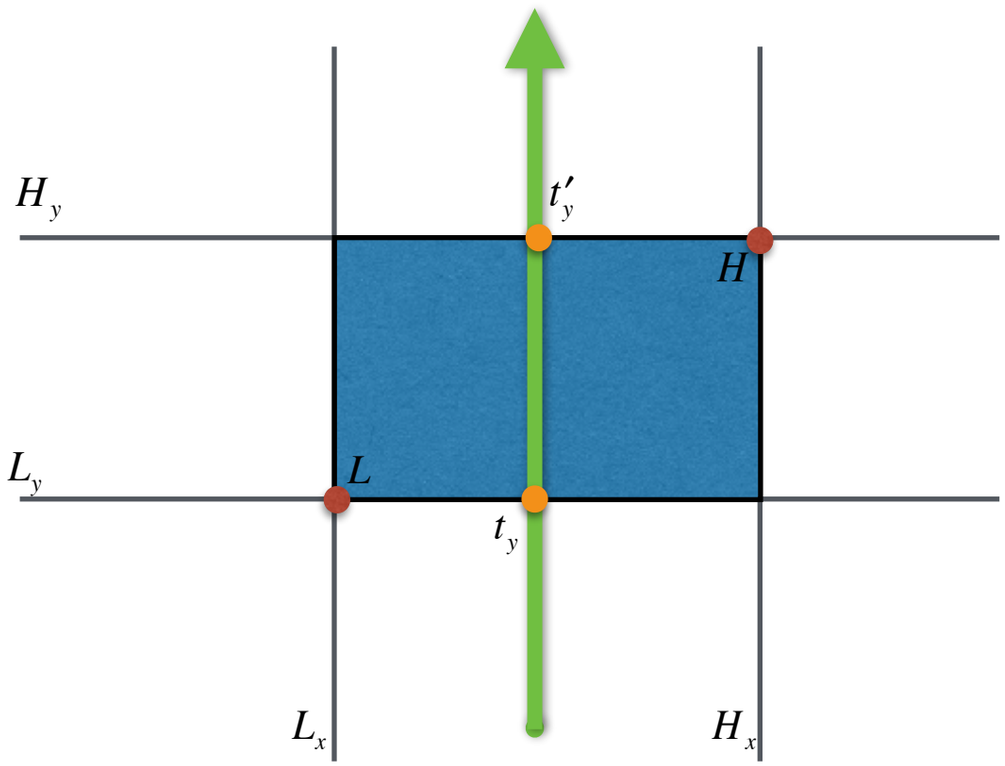
- likewise for y coordinate

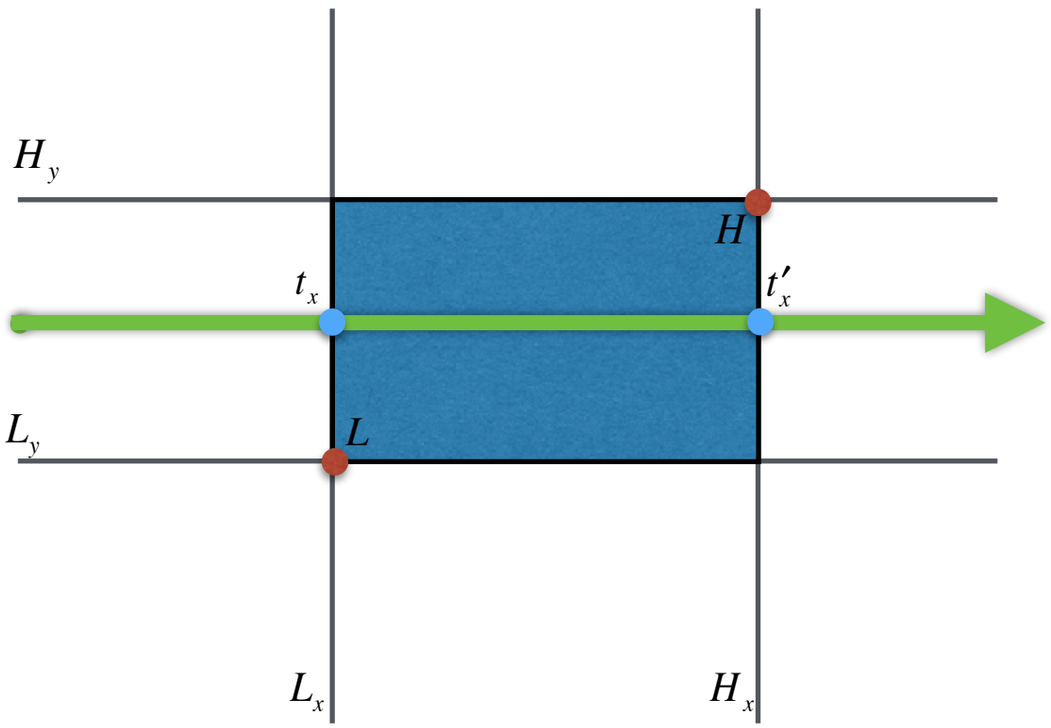




3D Space & Corner Cases

- in 3D space:
  - t_z, t'_z computed analogously
  - check $\max(t_x, t_y, t_z) < \min(t'_x, t'_y, t'_z), \min(t'_x, t'_y, t'_z) > 0$
- corner cases ($d_x=0$ or $d_y=0$ or $d_z=0$)





The algorithm

```
if dx >= 0
   tx  = (Lx - ox)/dx
   tx' = (Hx - ox)/dx
else
   tx  = (Hx - ox)/dx
   tx' = (Lx - ox)/dx
```

this computes tx, tx’

ty, ty’ and tz, tz’ are computed analogously

```
t  = max(tx ,ty ,tz )
t’ = min(tx’,ty’,tz’)
```

t = distance to enter box

t’ = distance to exit box

```
t < t’ && t’ > 0
```

check for hit

Corner cases are covered by IEEE floating-point arithmetic! (1 / 0 = Infinity, -1 / 0 = -Infinity)

IT UNIVERSITY OF COPENHAGEN

Questions

IT UNIVERSITY OF COPENHAGEN

kd-Trees

The Problem

- naive ray tracing: check hit function of every object in a scene
- very expensive if we have many objects in a scene
- infeasible for triangle meshes with millions of triangles

Solution

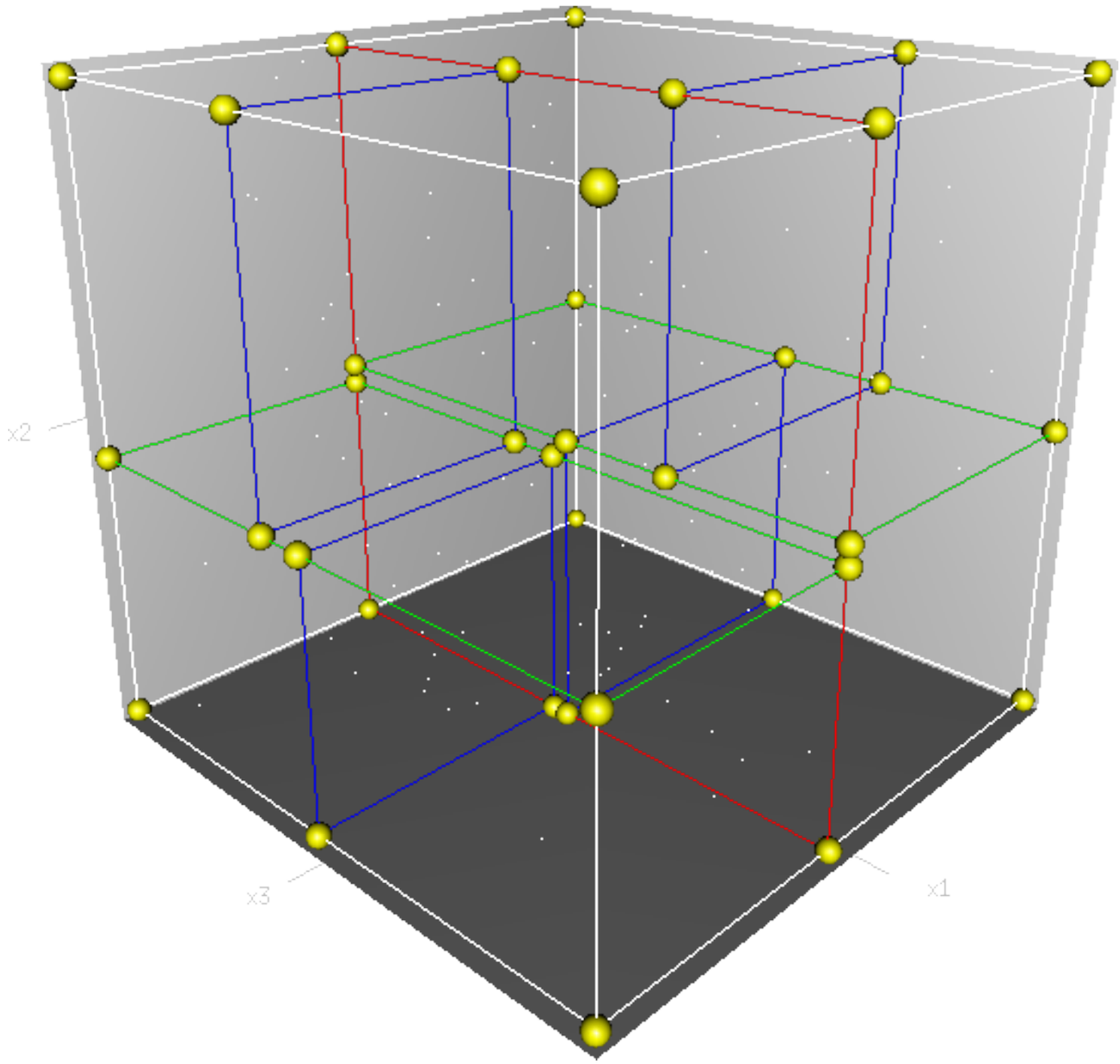
A. in advance:

   1. divide the scene into zones
   2. calculate which objects are in which zone

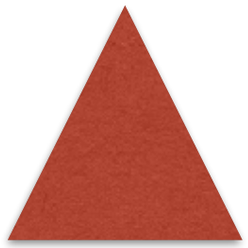
B. at ray tracing time

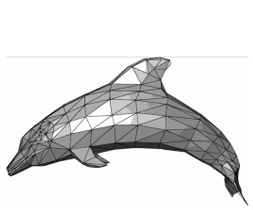
   1. find the zones which the ray passes
   2. only check intersection with objects in those zones

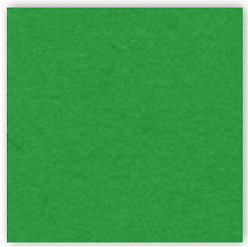
Example

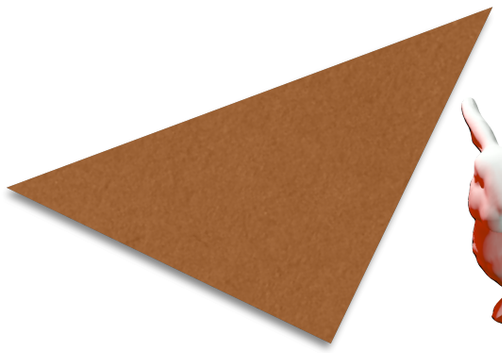


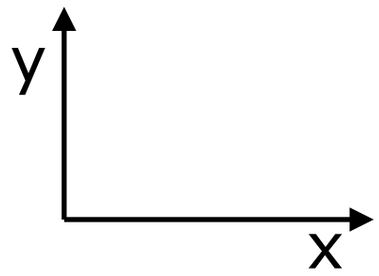
2D Example



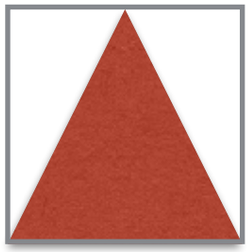


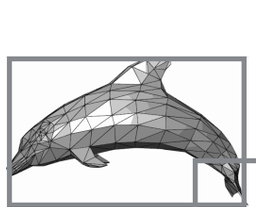


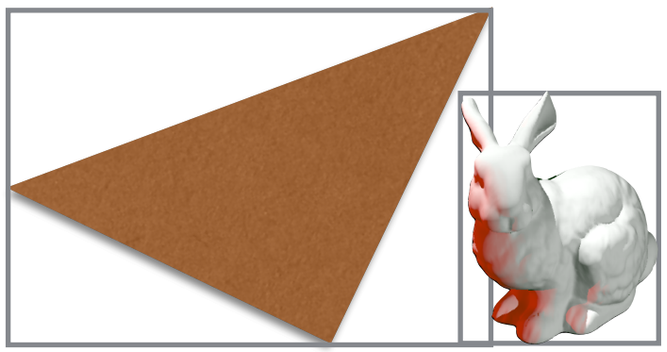


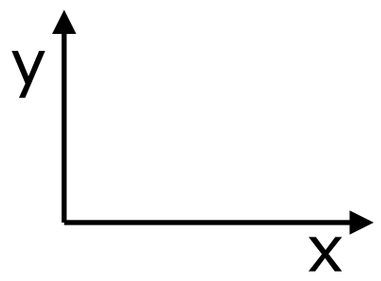


2D Example

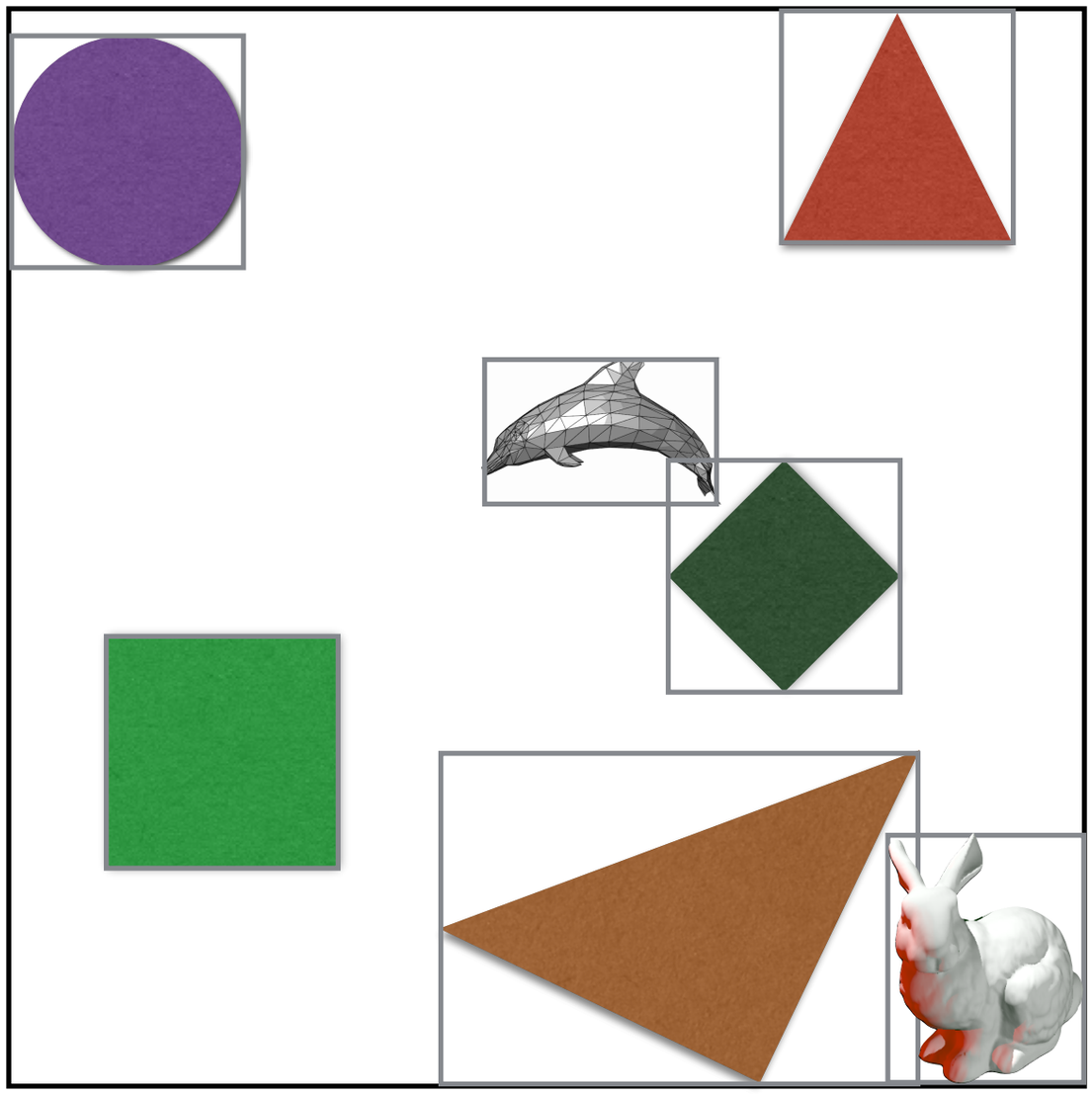


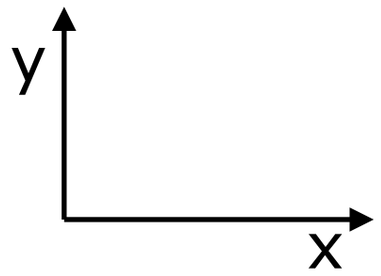






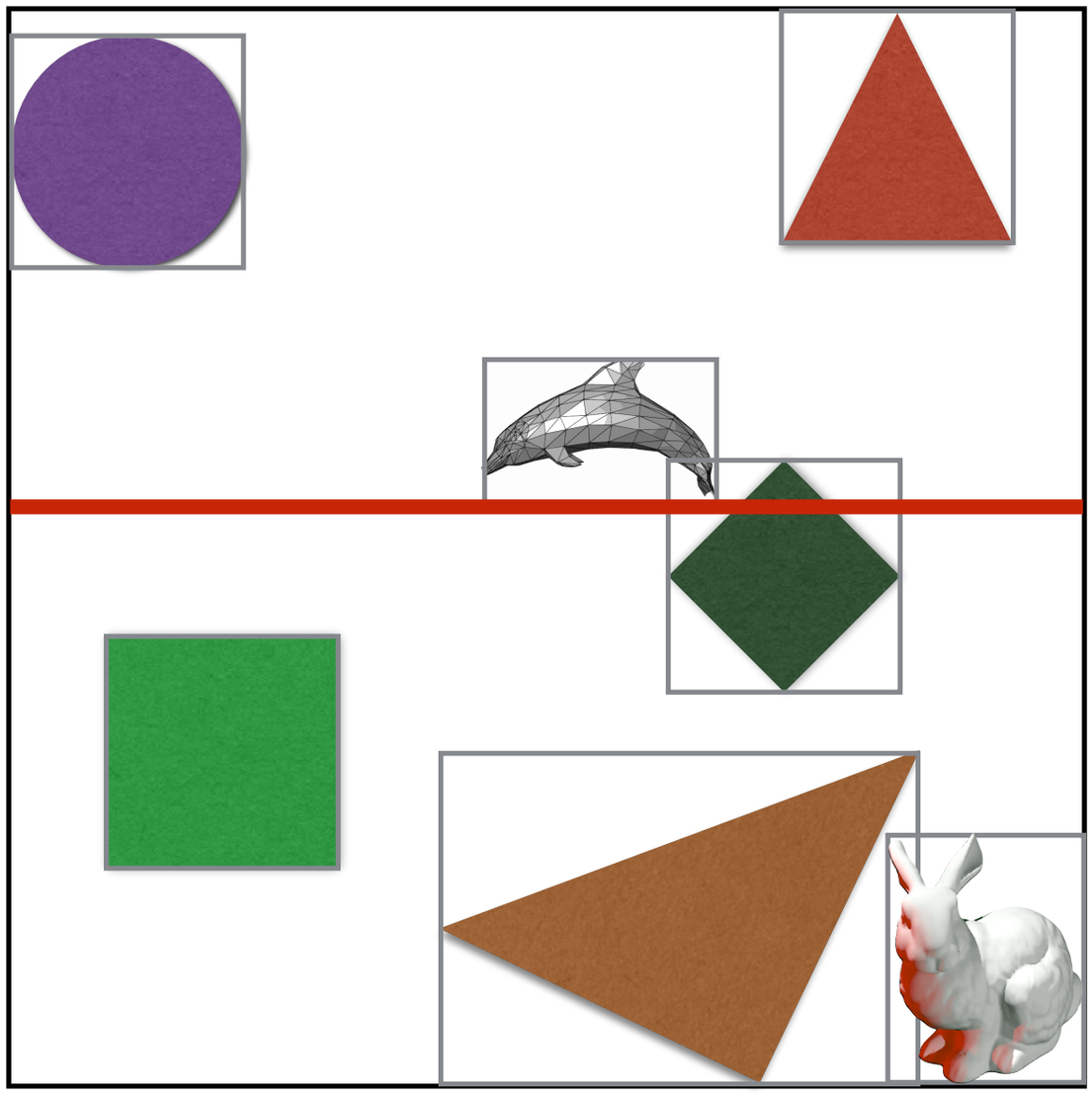
2D Example

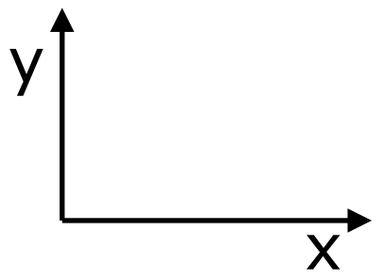




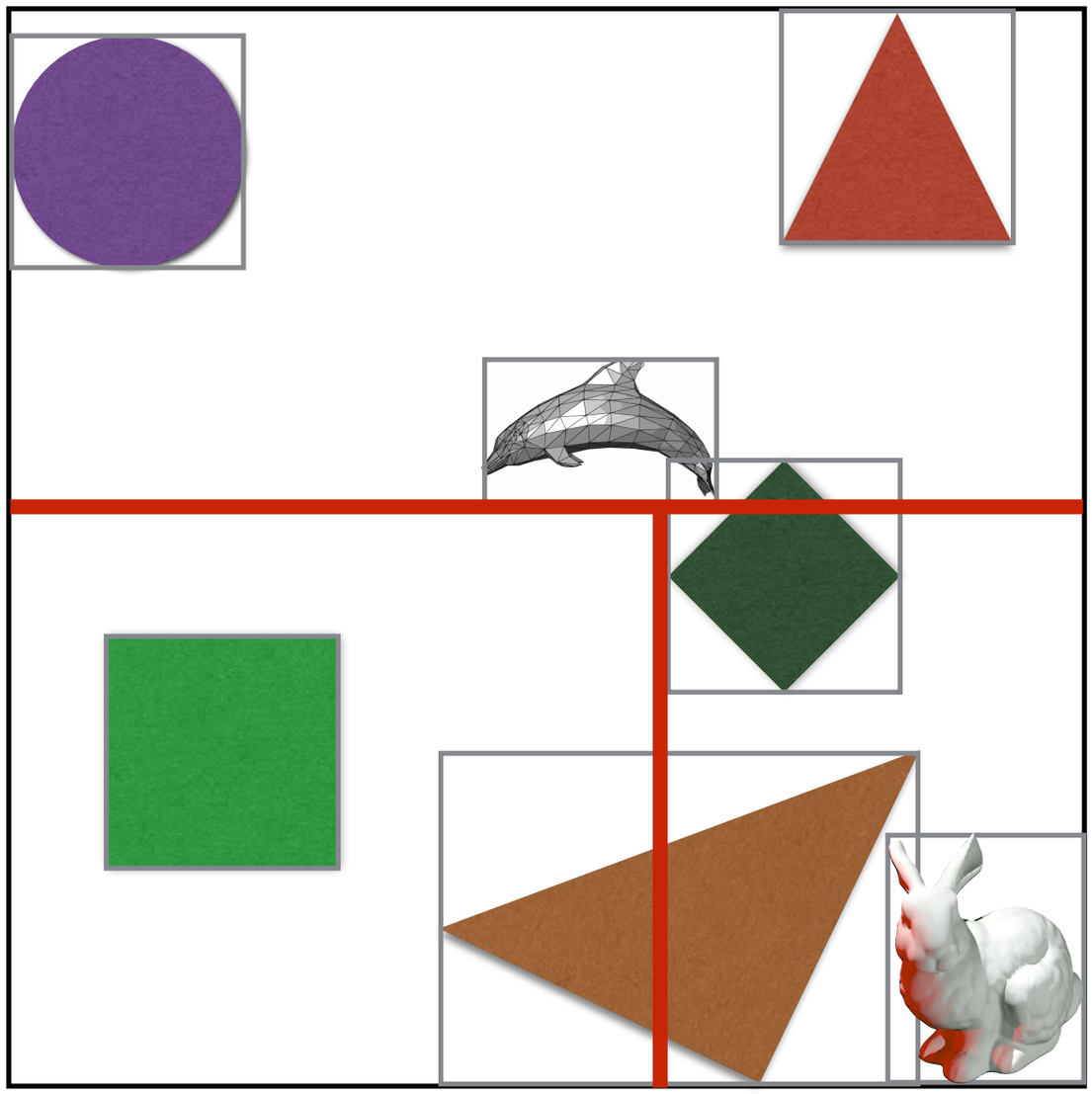
2D Example

y=5



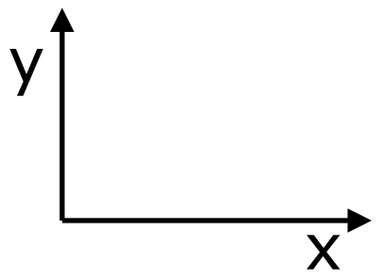


2D Example

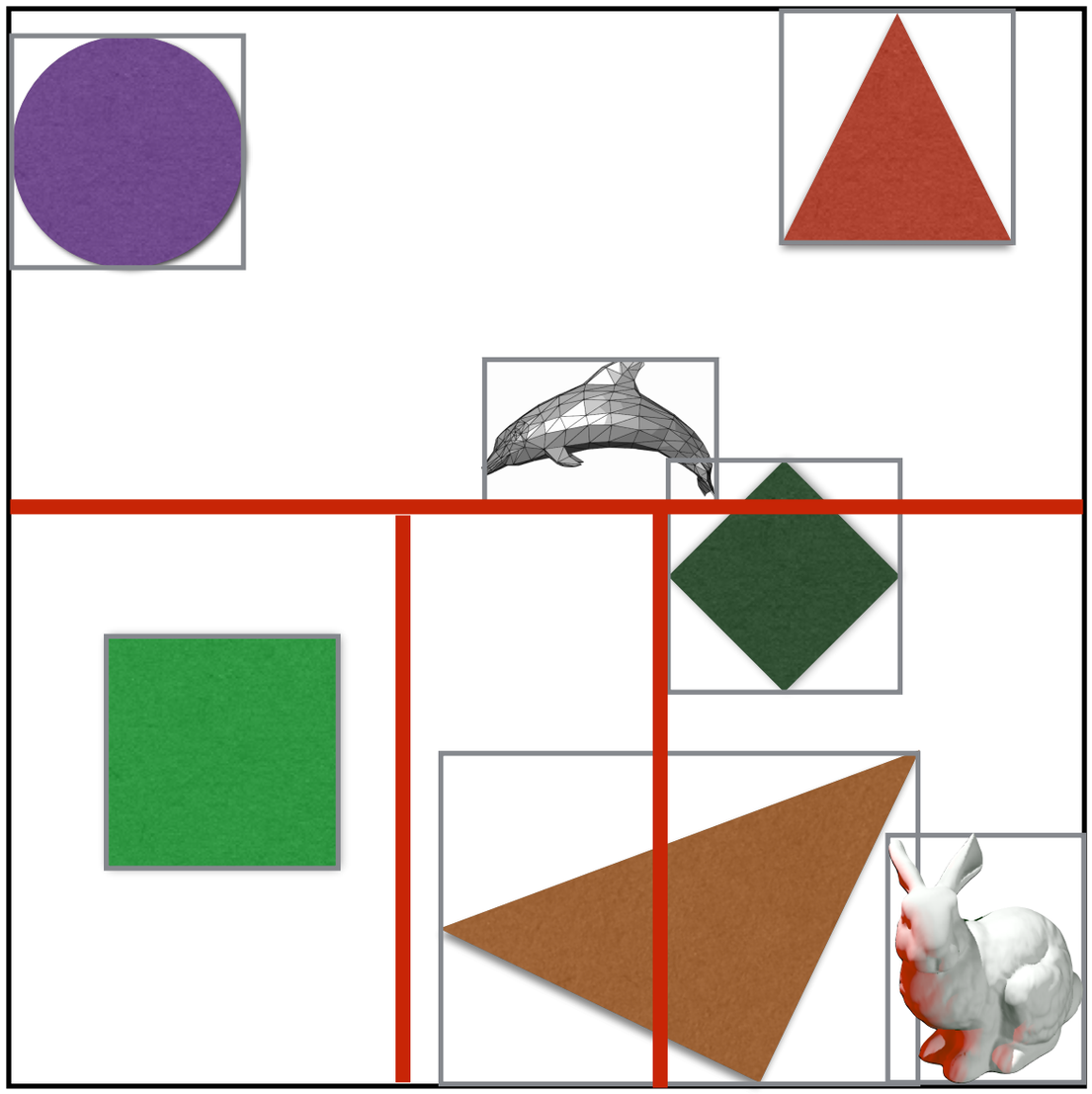


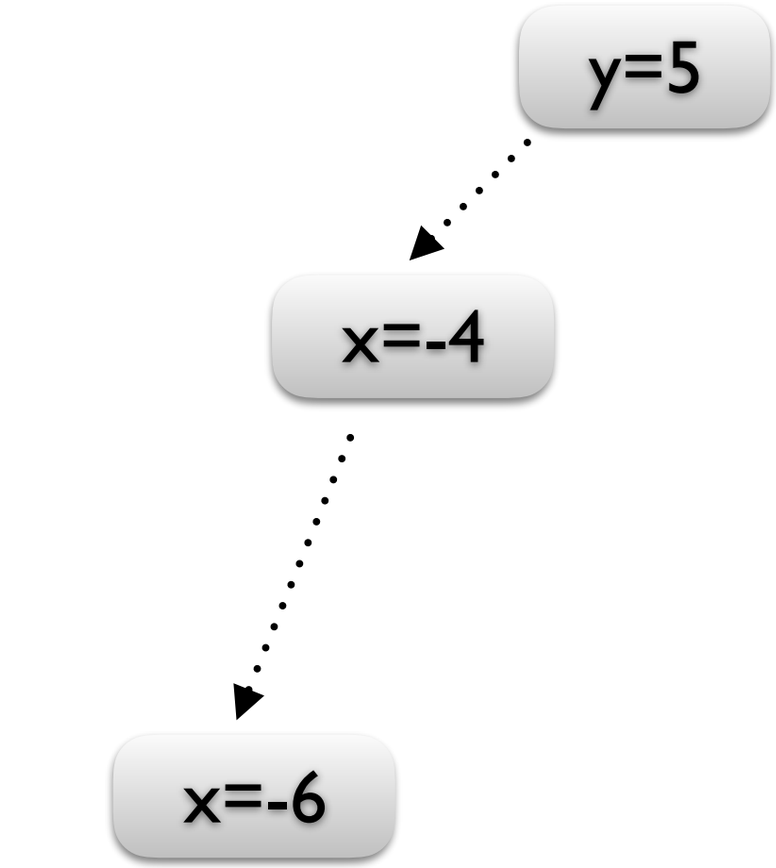
y=5

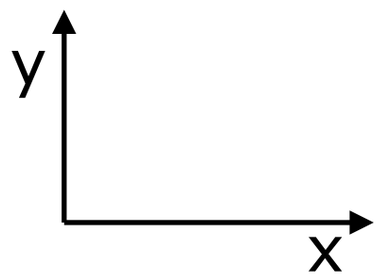
x=-4



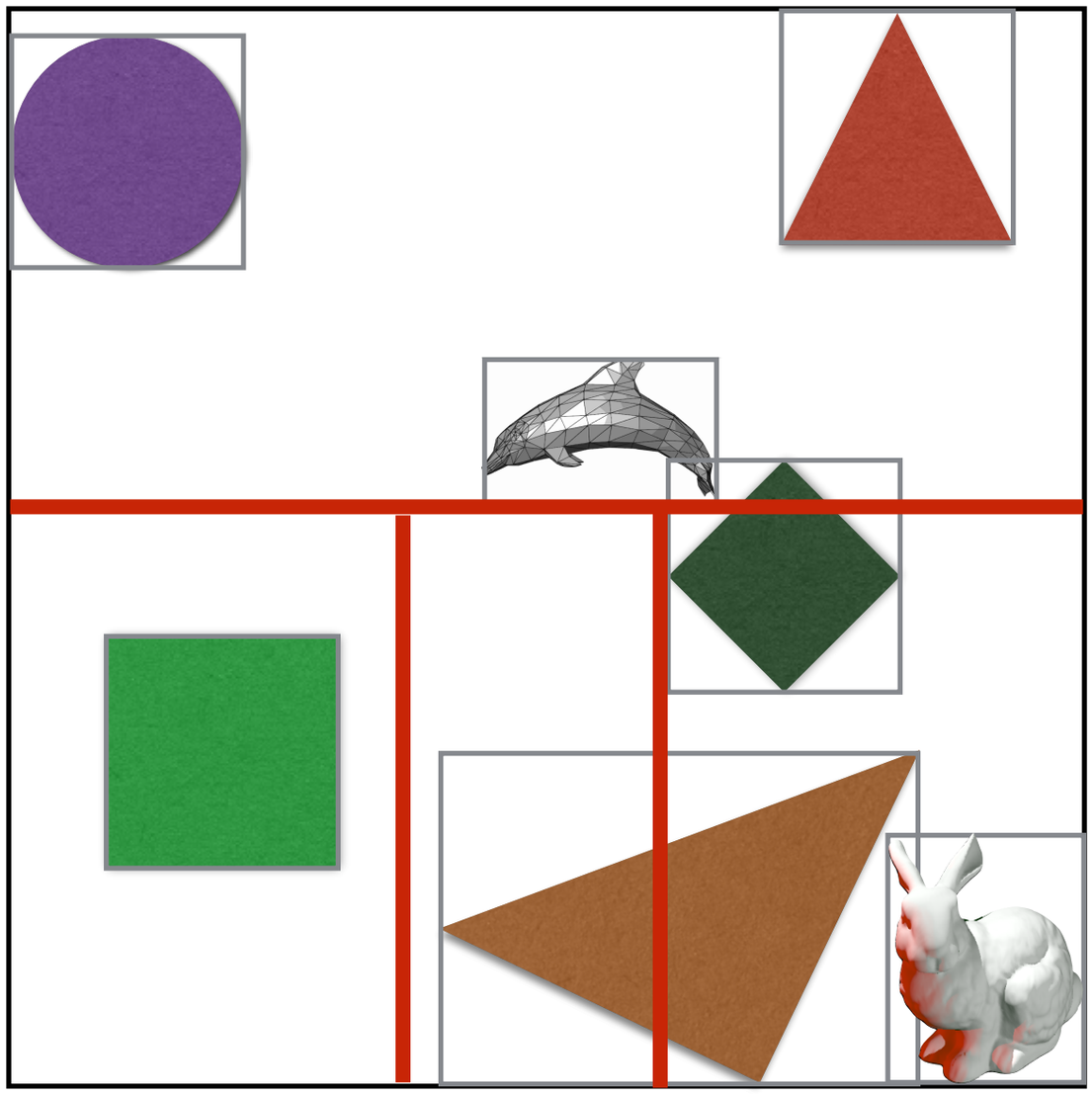
2D Example

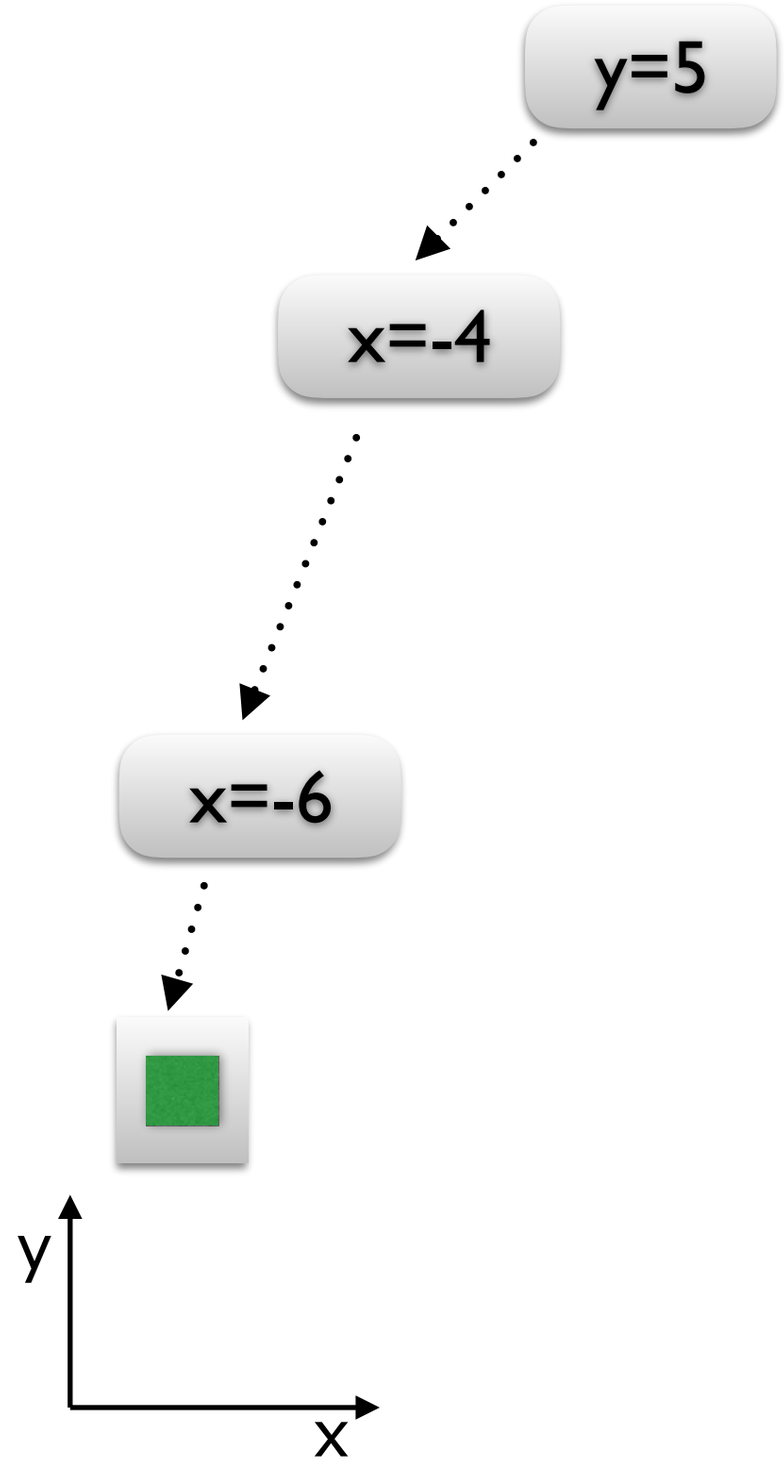




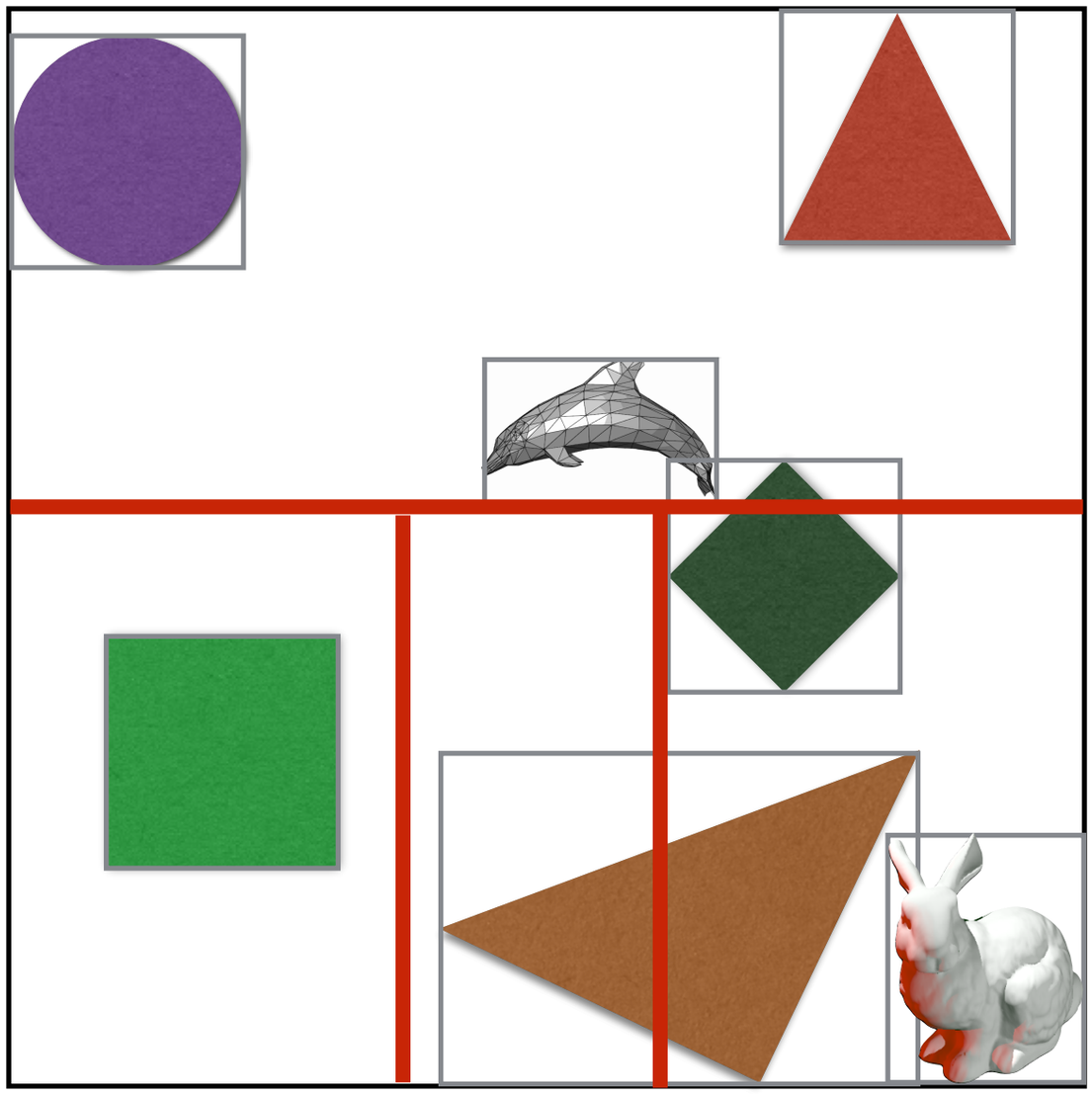


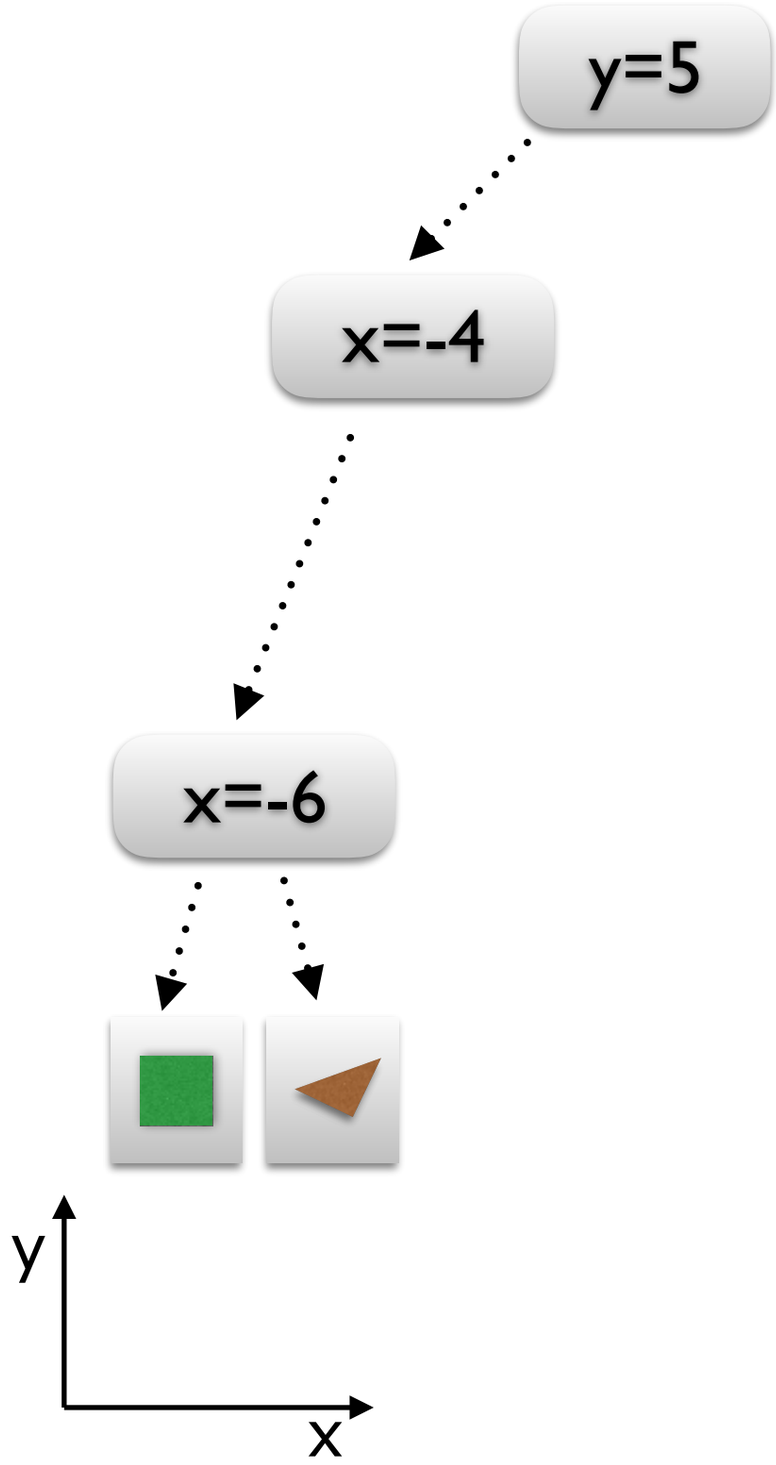
2D Example



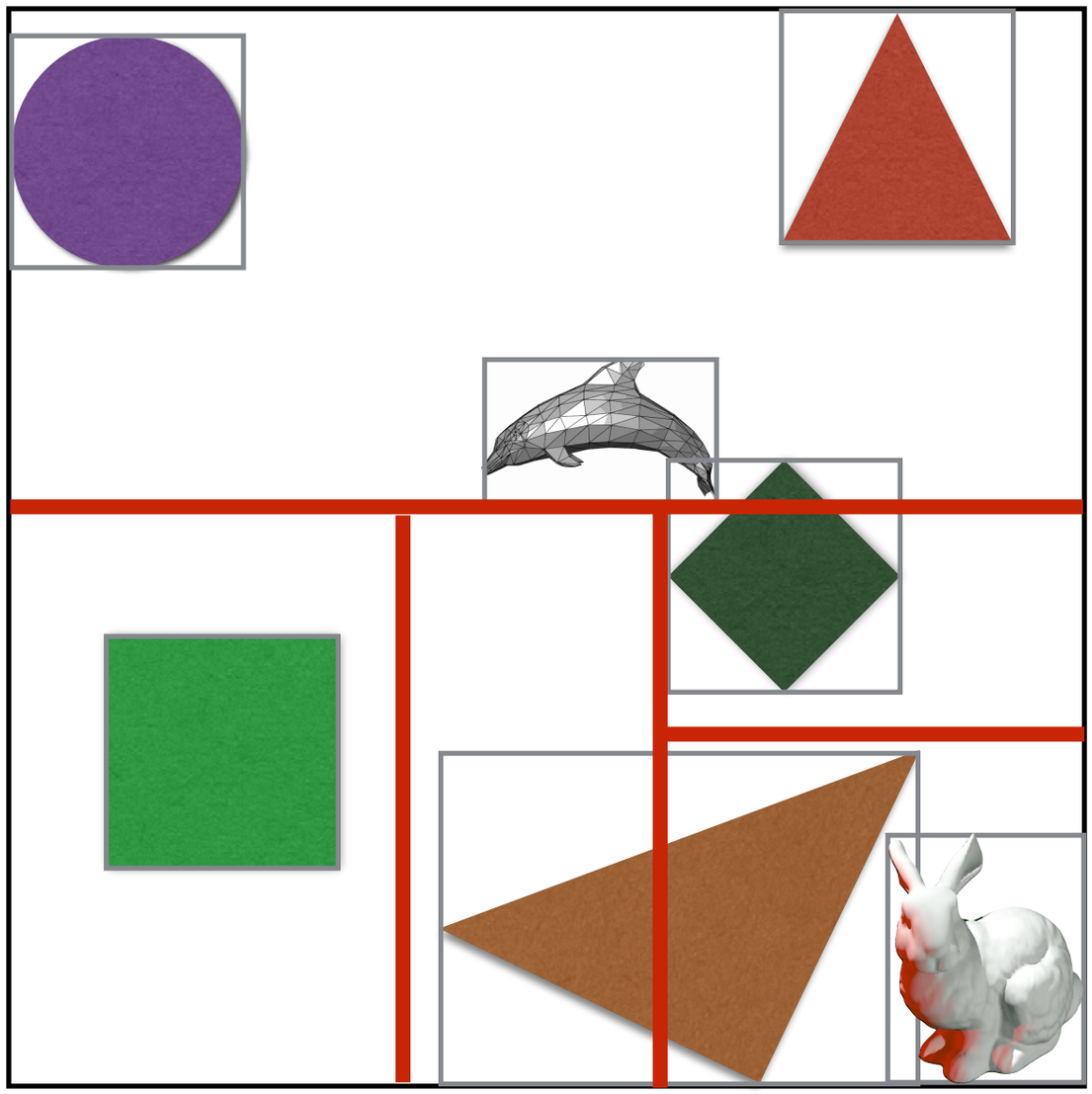


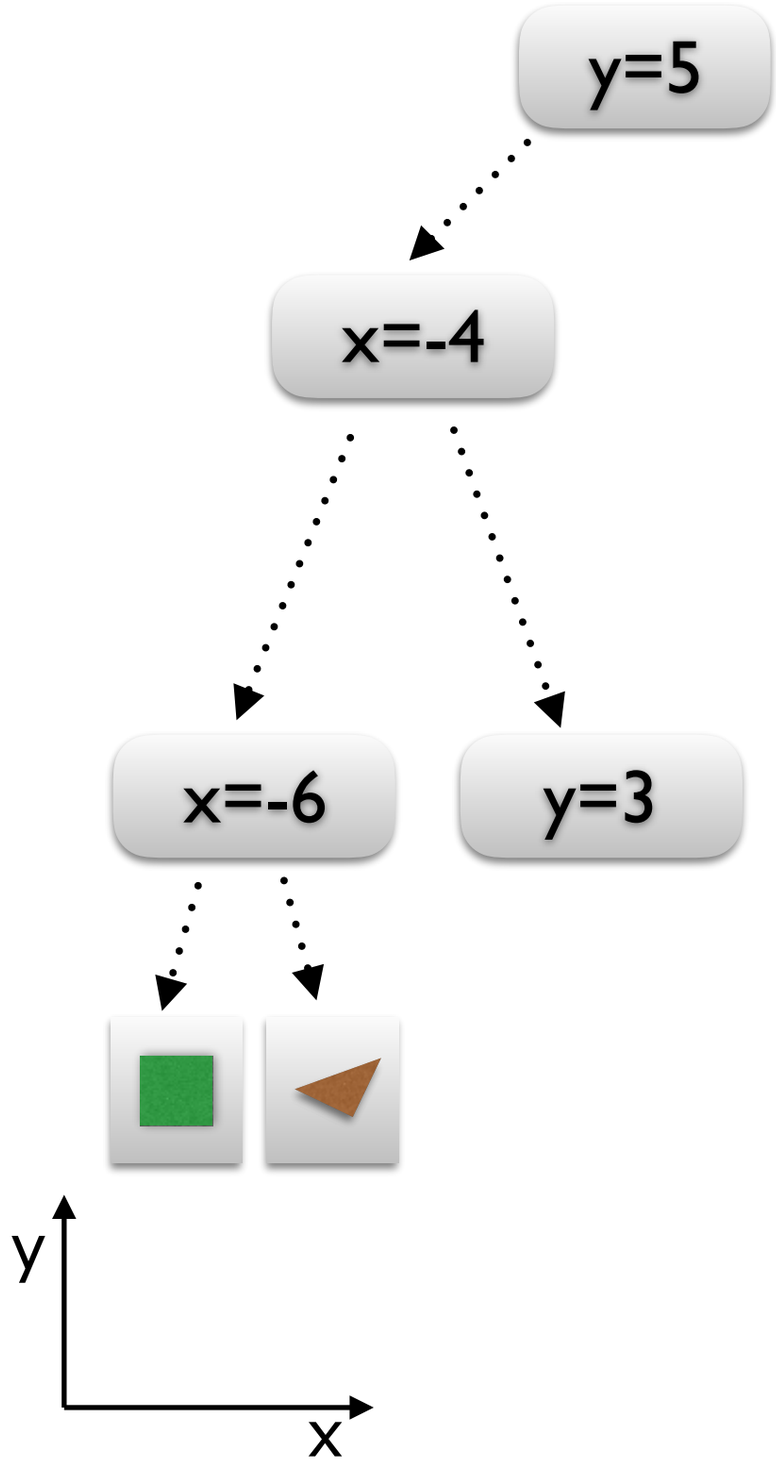
2D Example



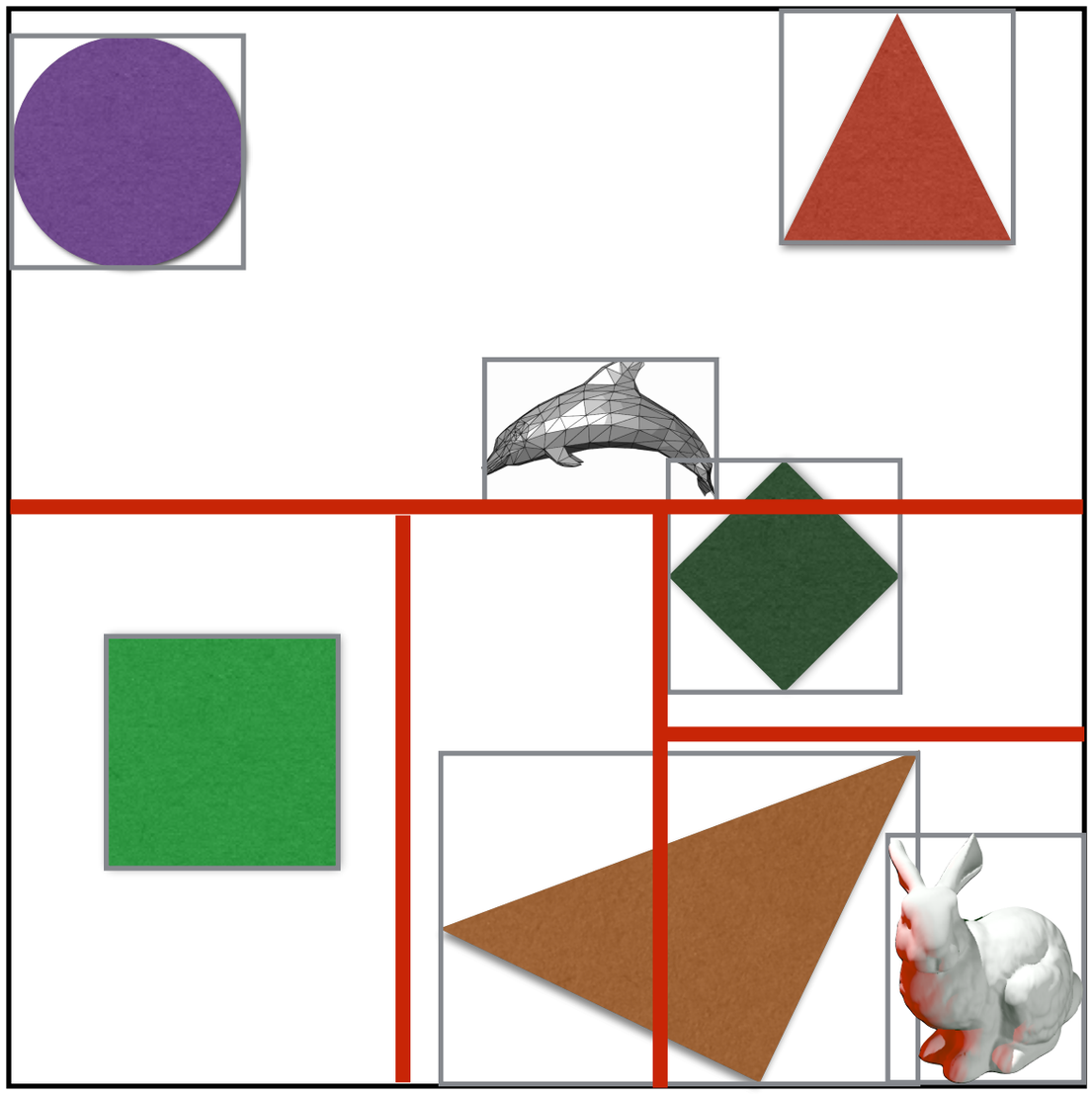


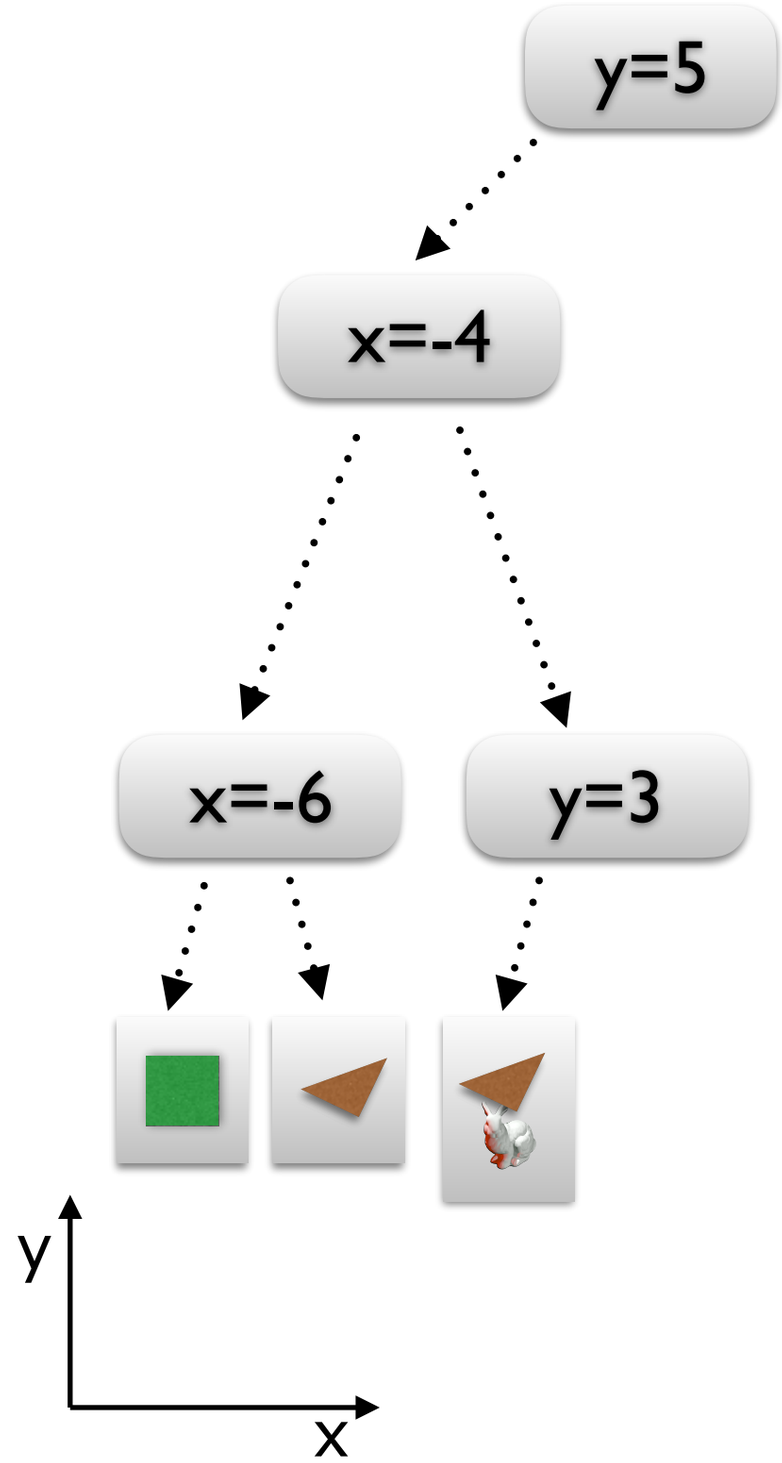
2D Example



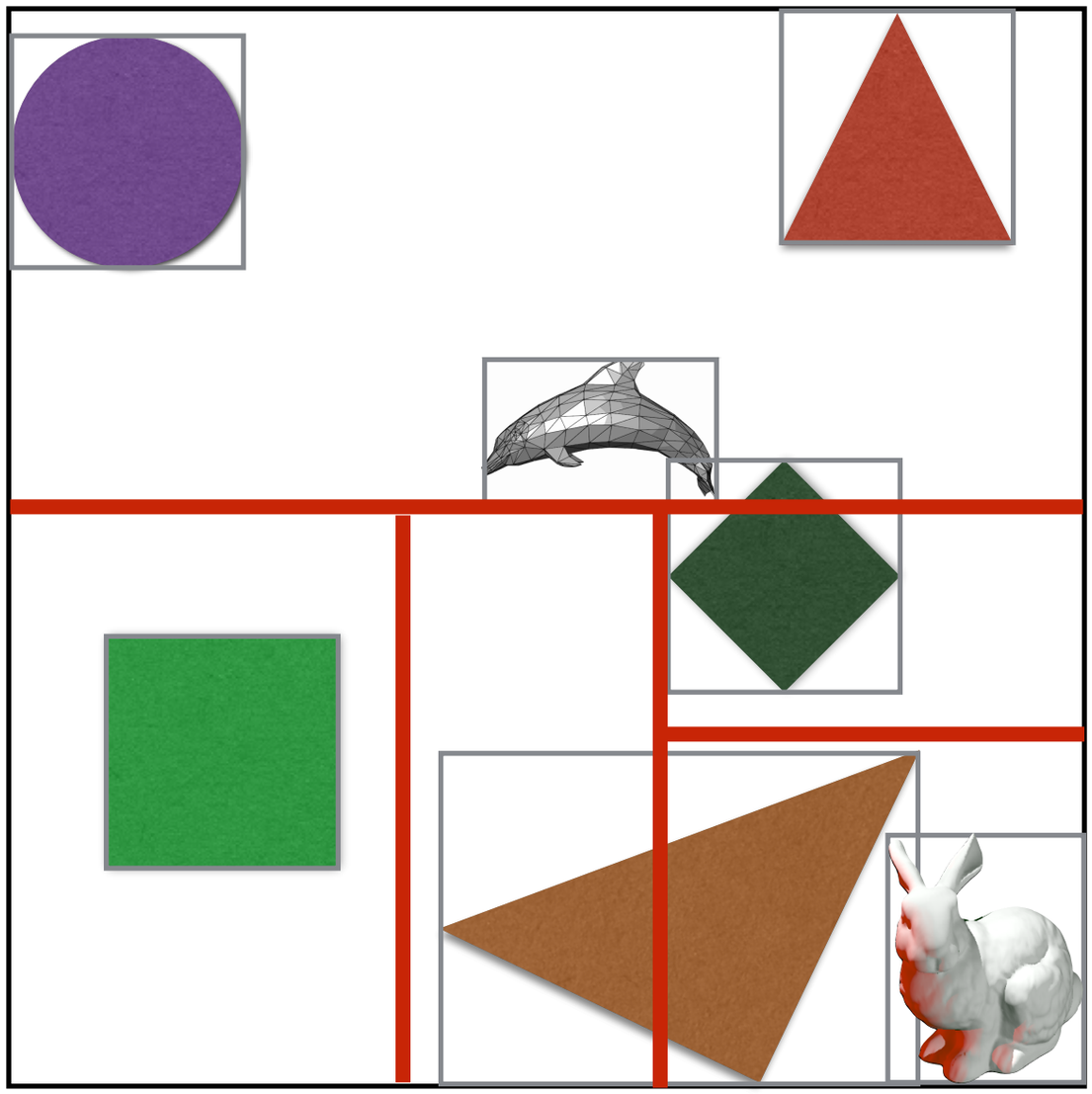


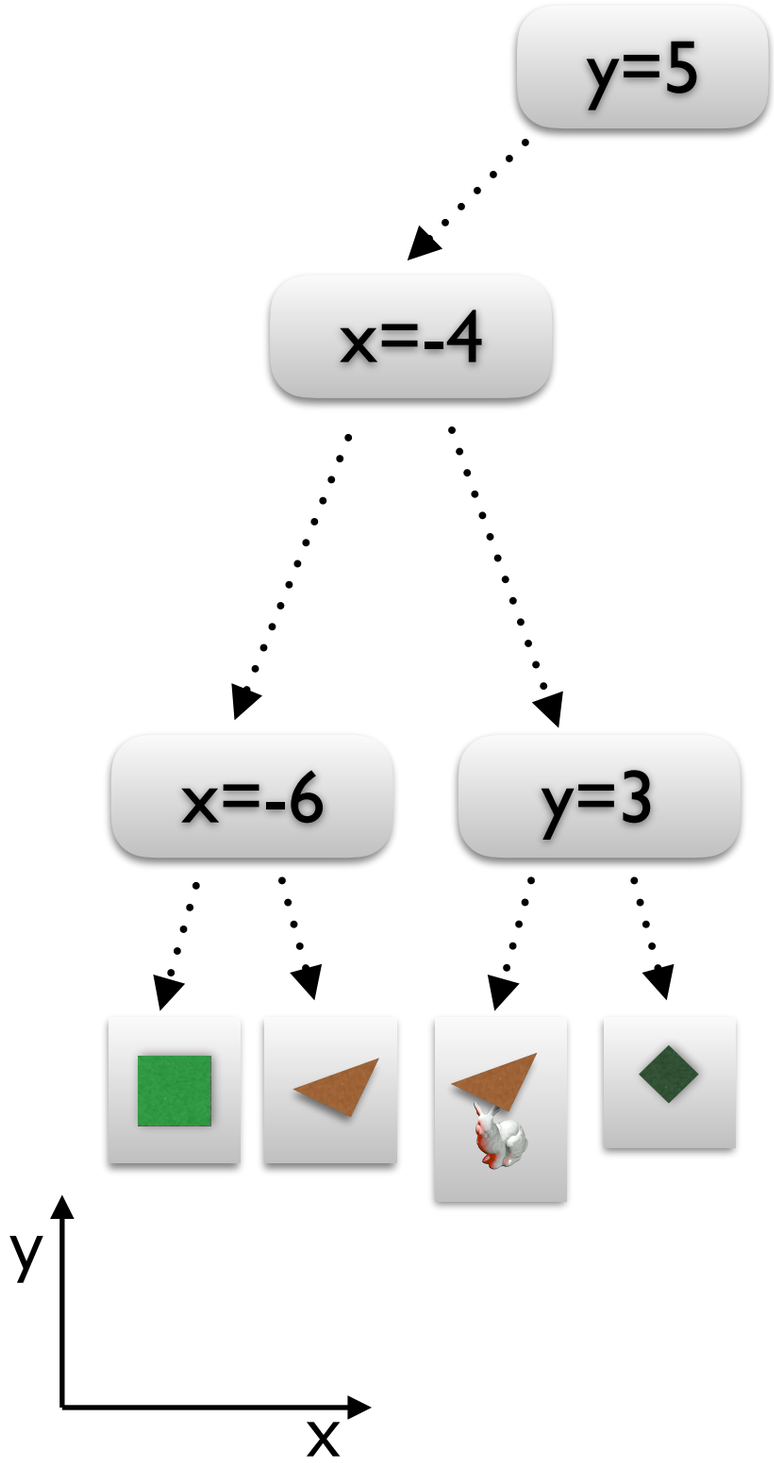
2D Example



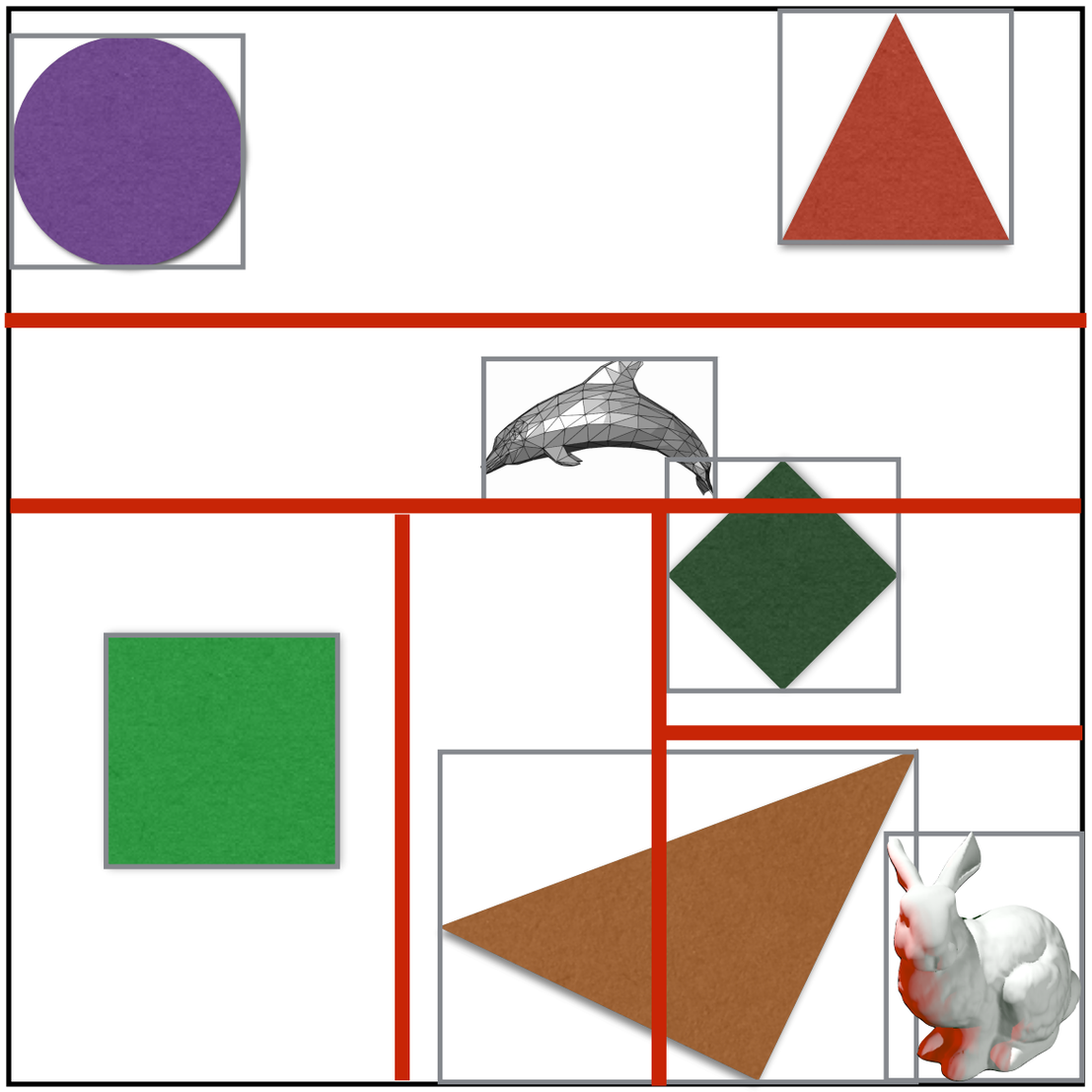


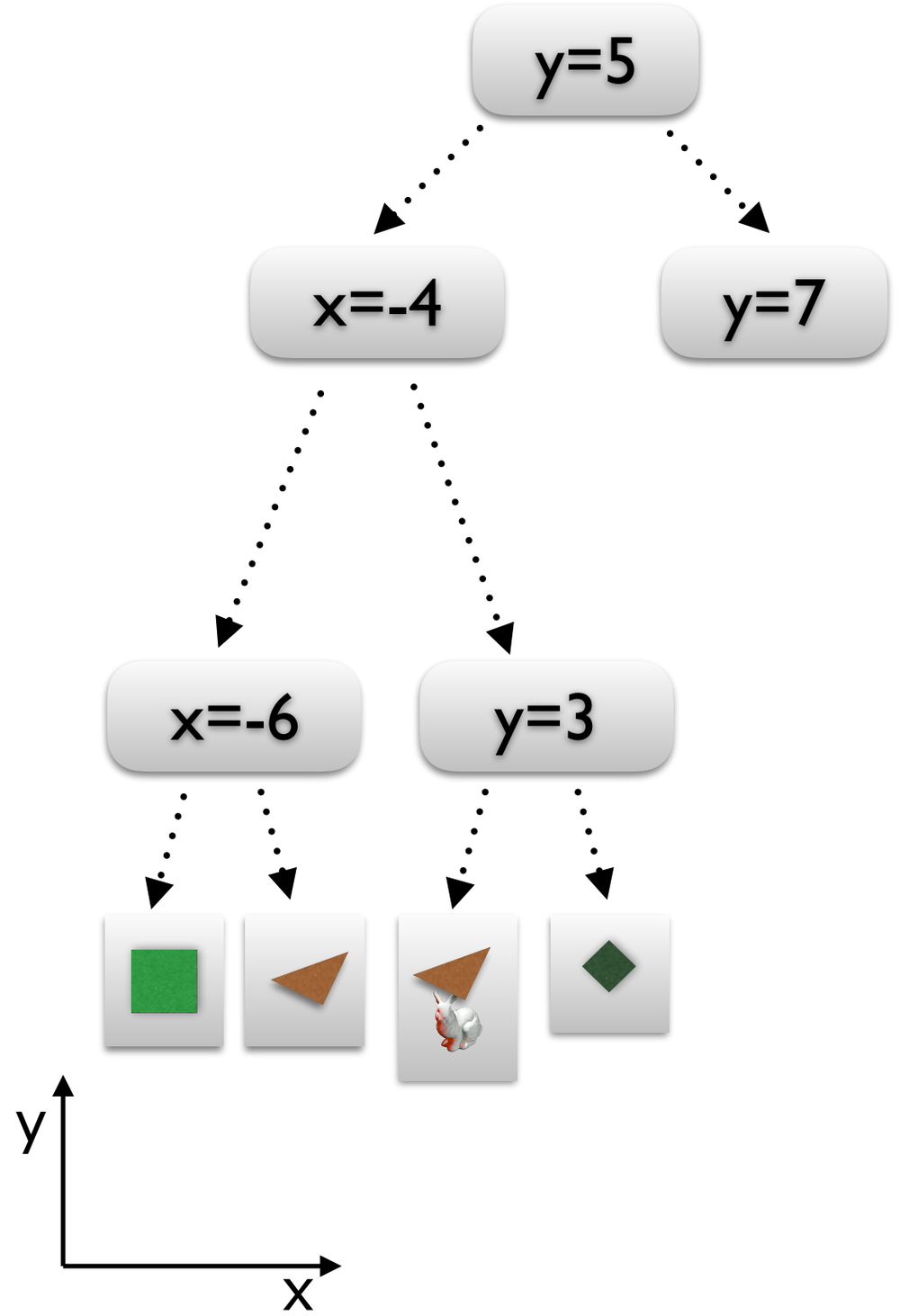
2D Example



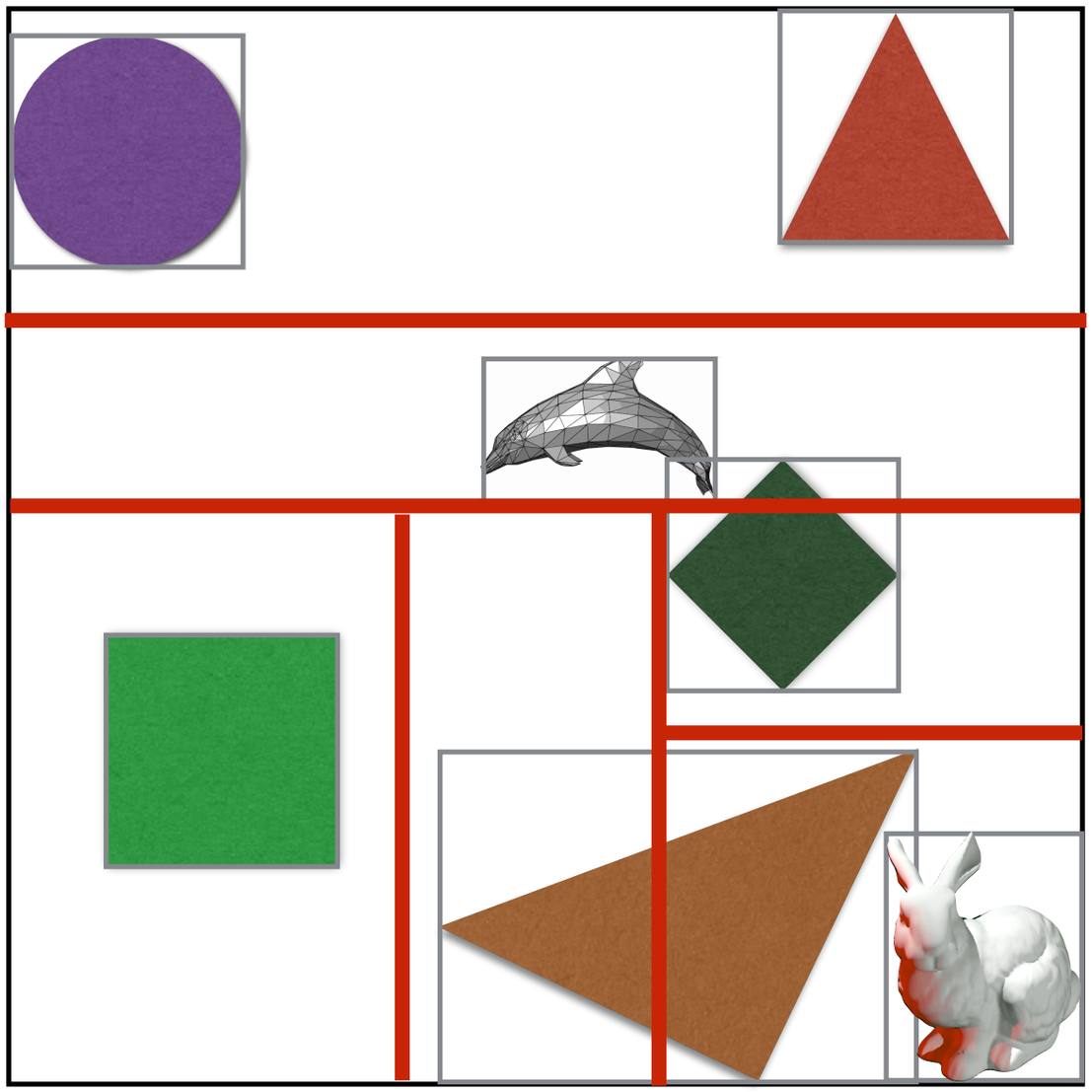


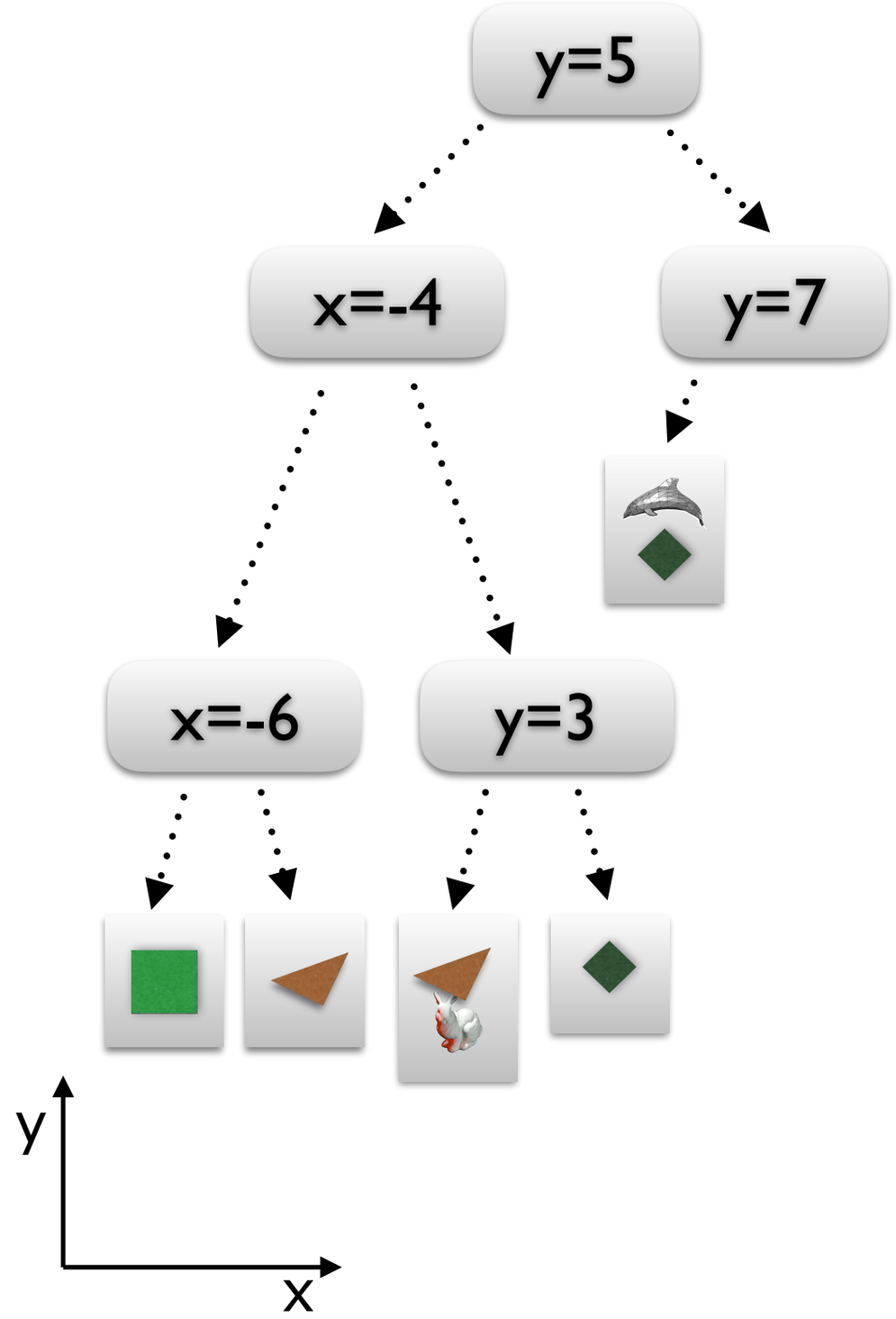
2D Example



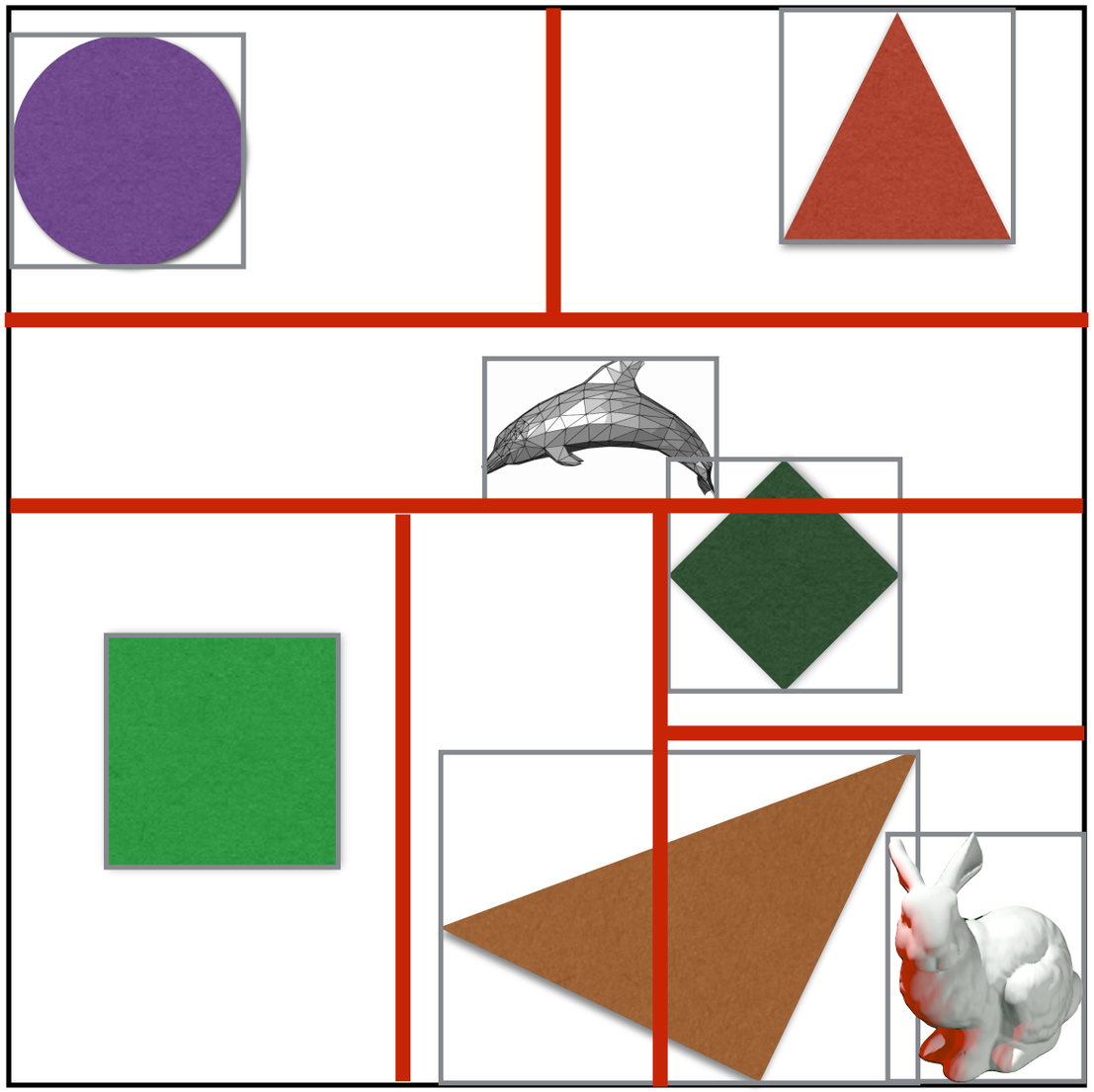


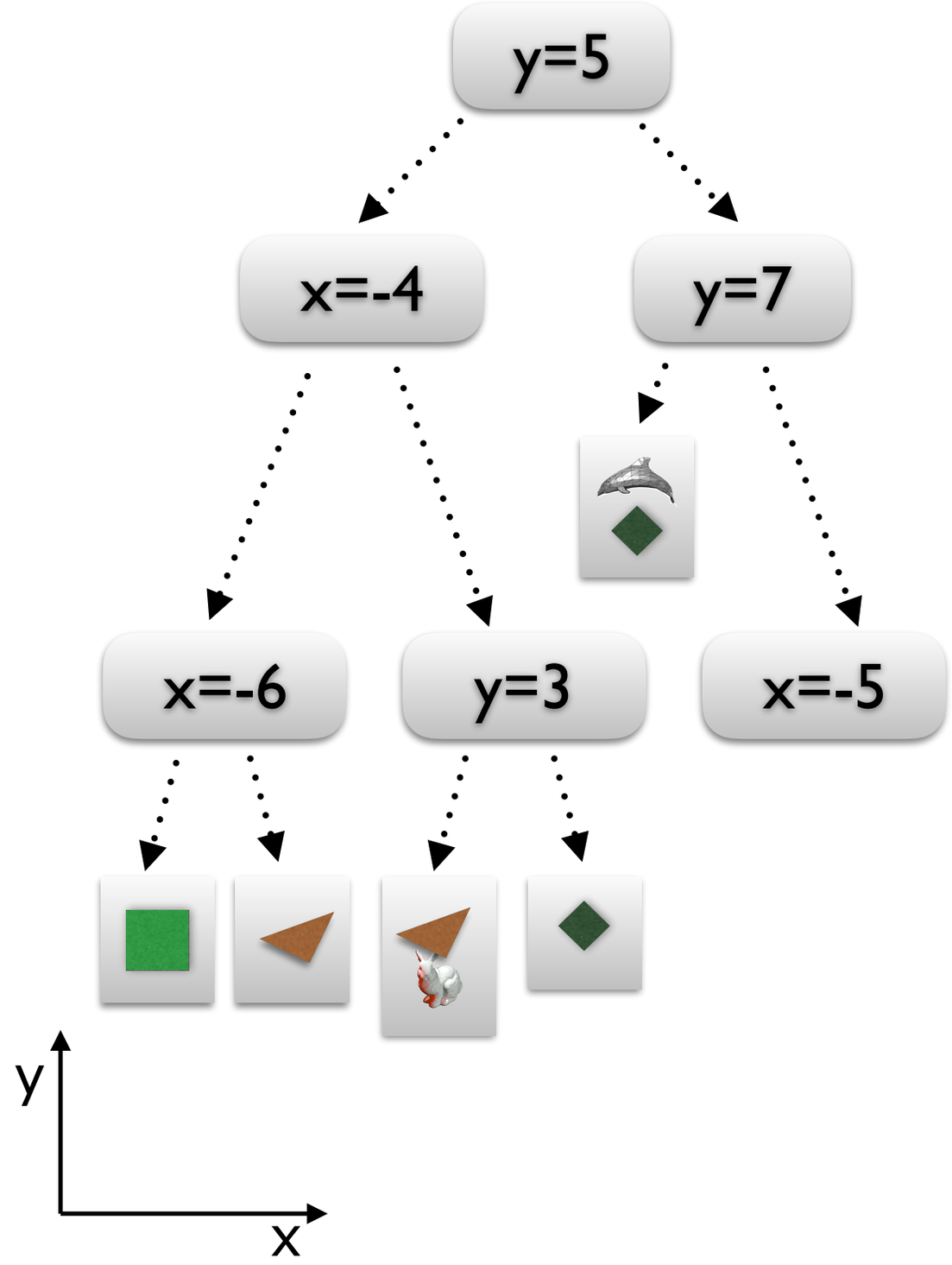
2D Example



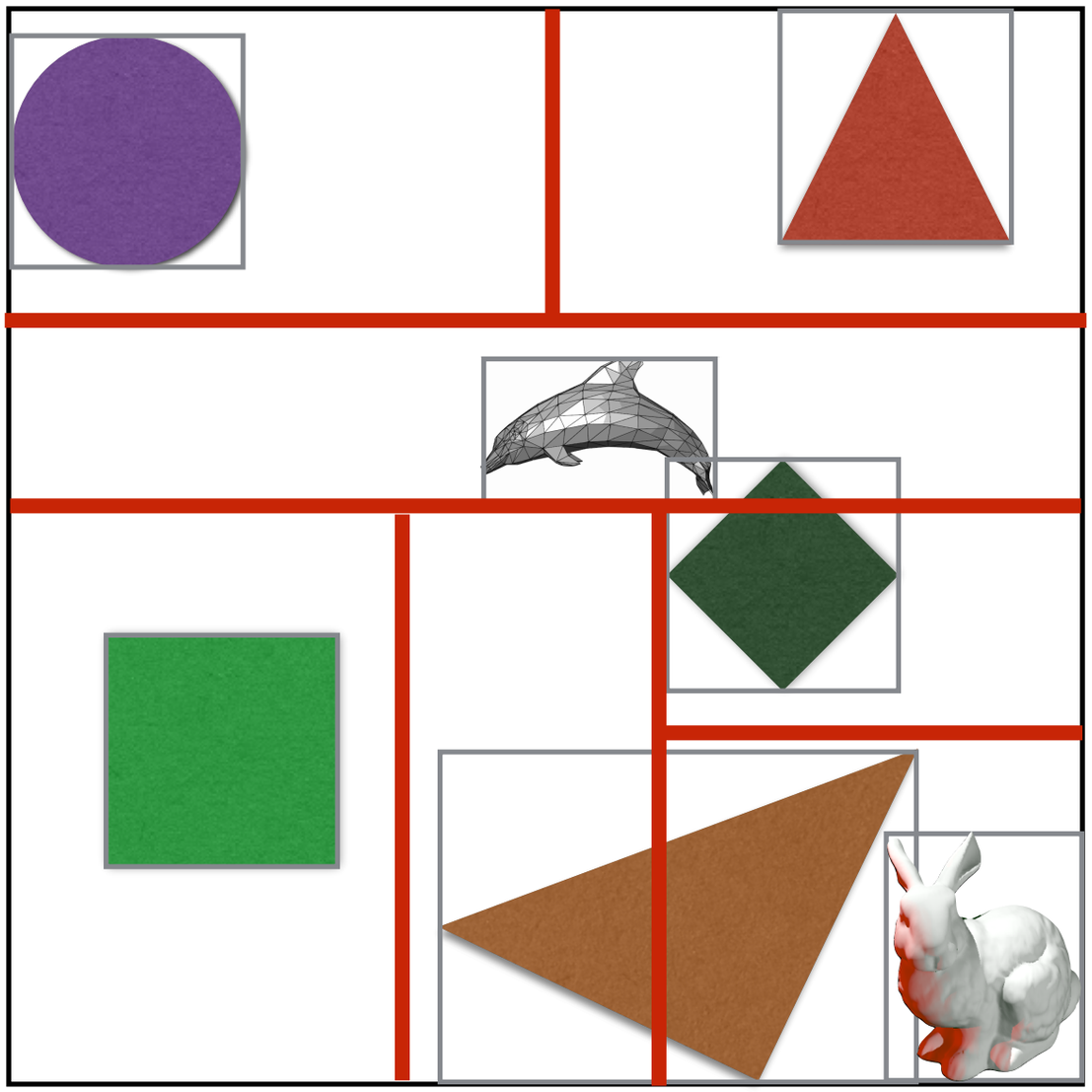


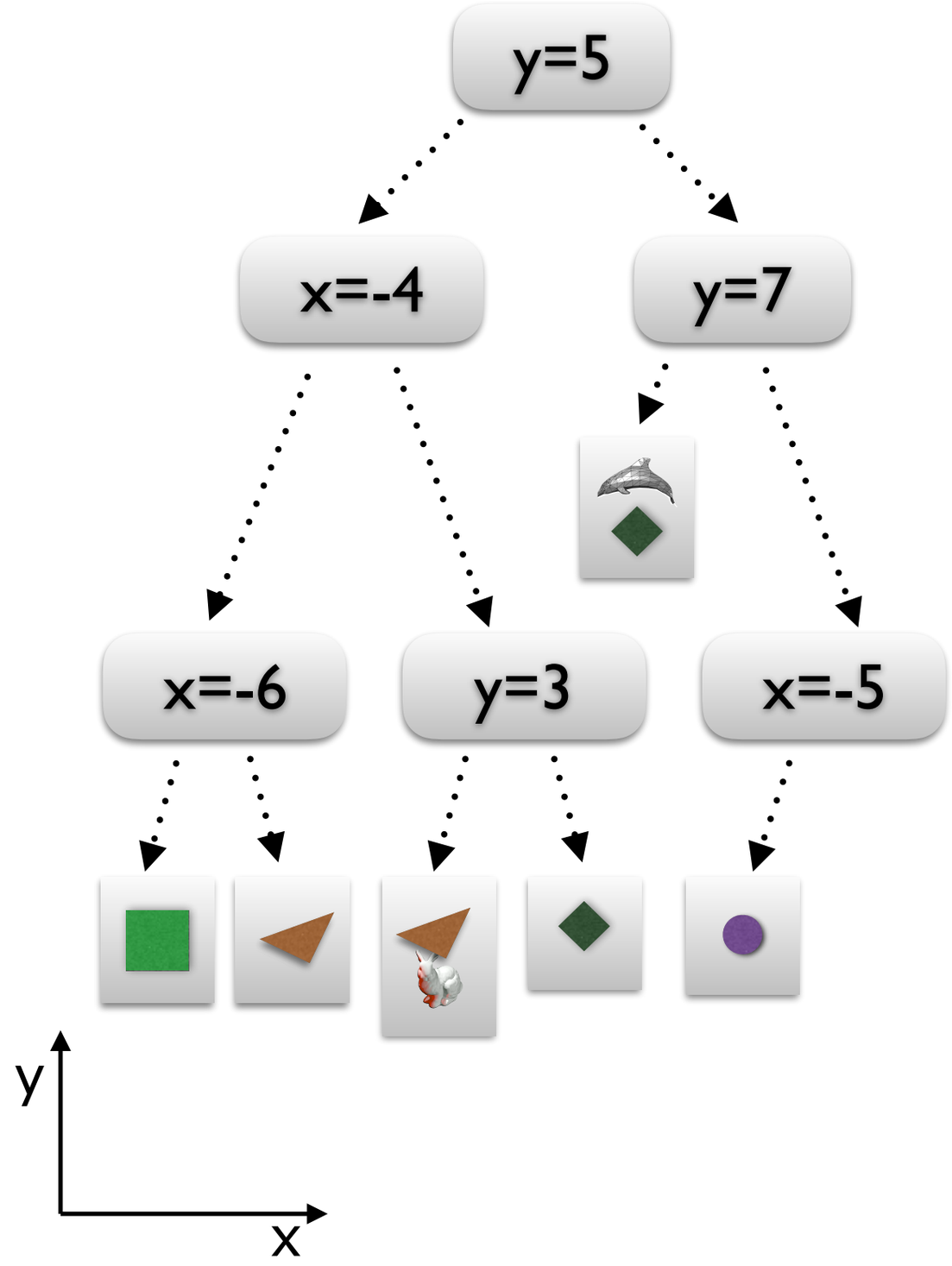
2D Example



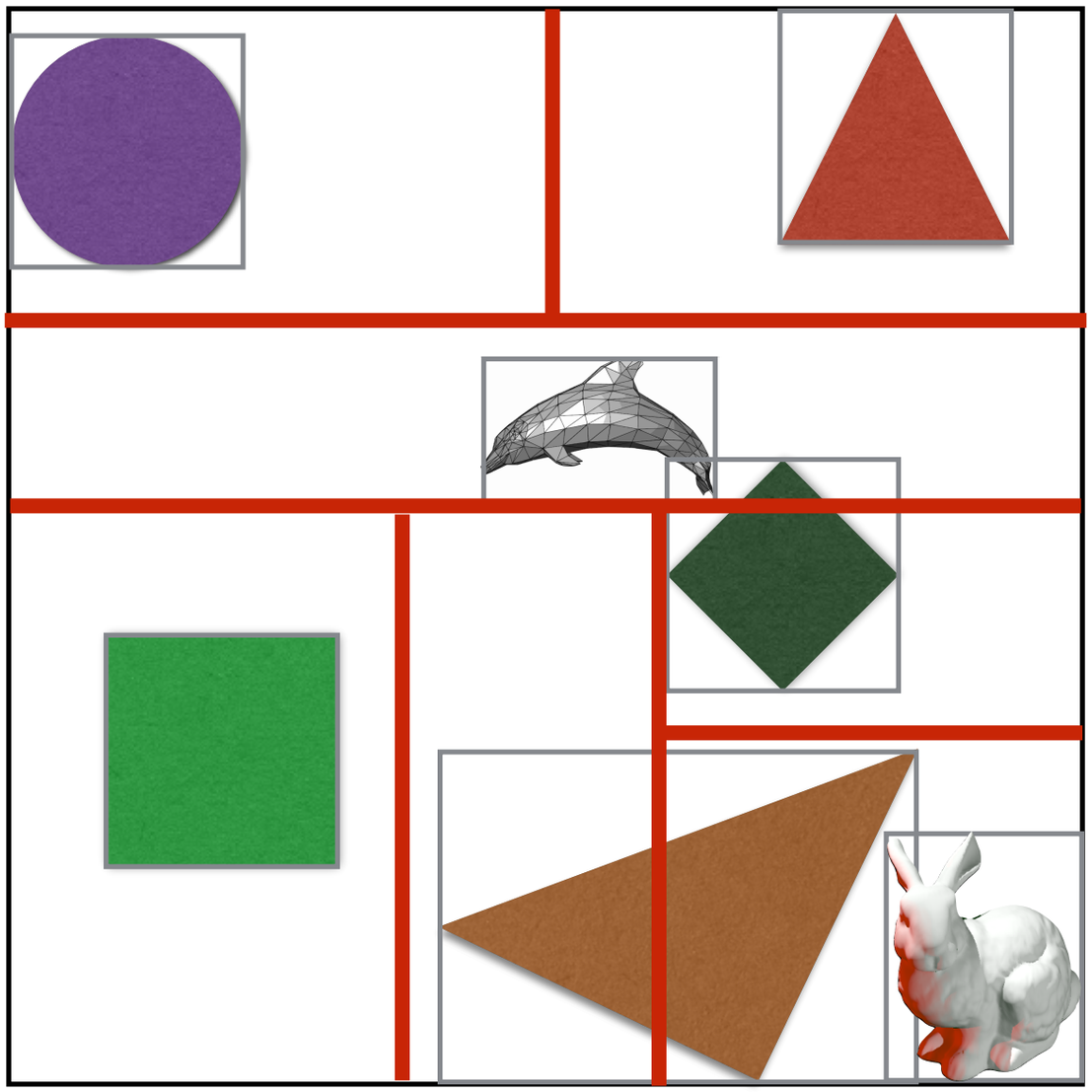


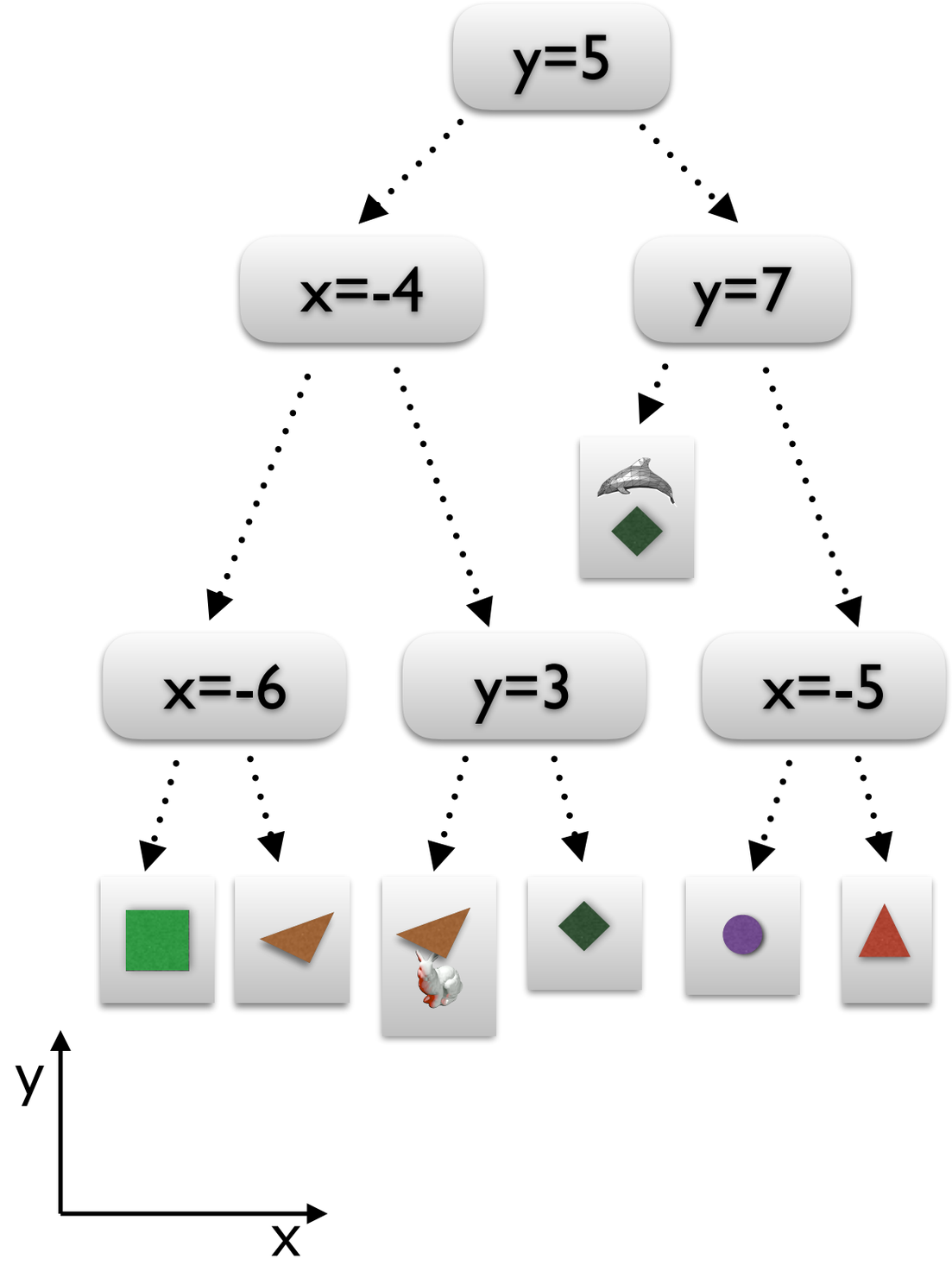
2D Example





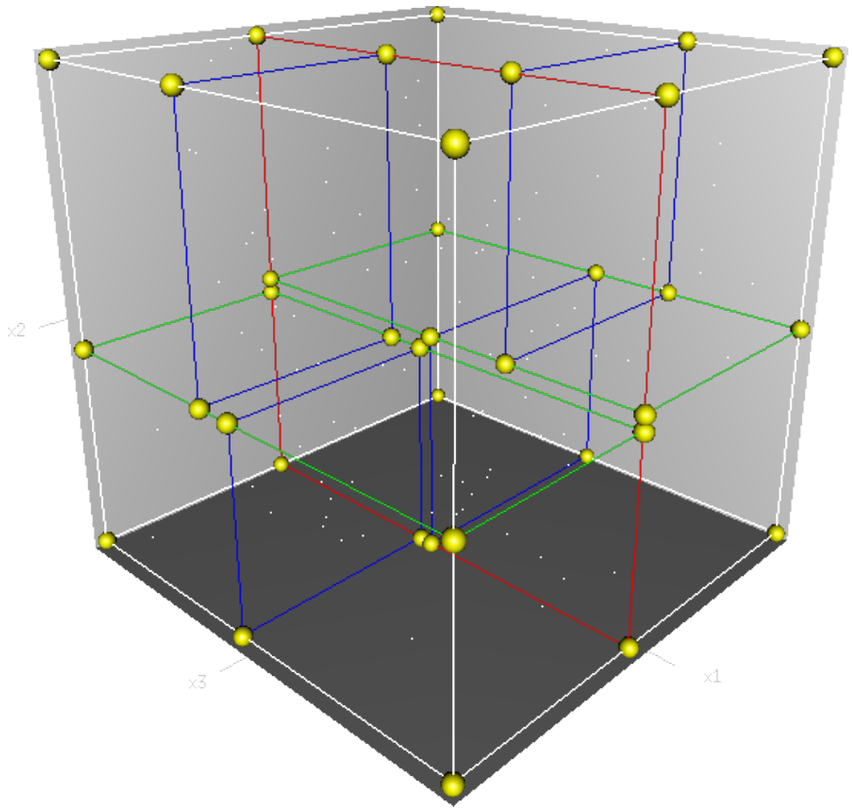
2D Example





In 3D

- same principle applies
- zones are divided by planes (instead of lines)



Recap

1. Calculate bounding box of each shape (note: some shapes don't have one)
2. Calculate bounding box for the kd-tree
3. Construct kd-tree by dividing the bounding box step-by-step

How to divide space?

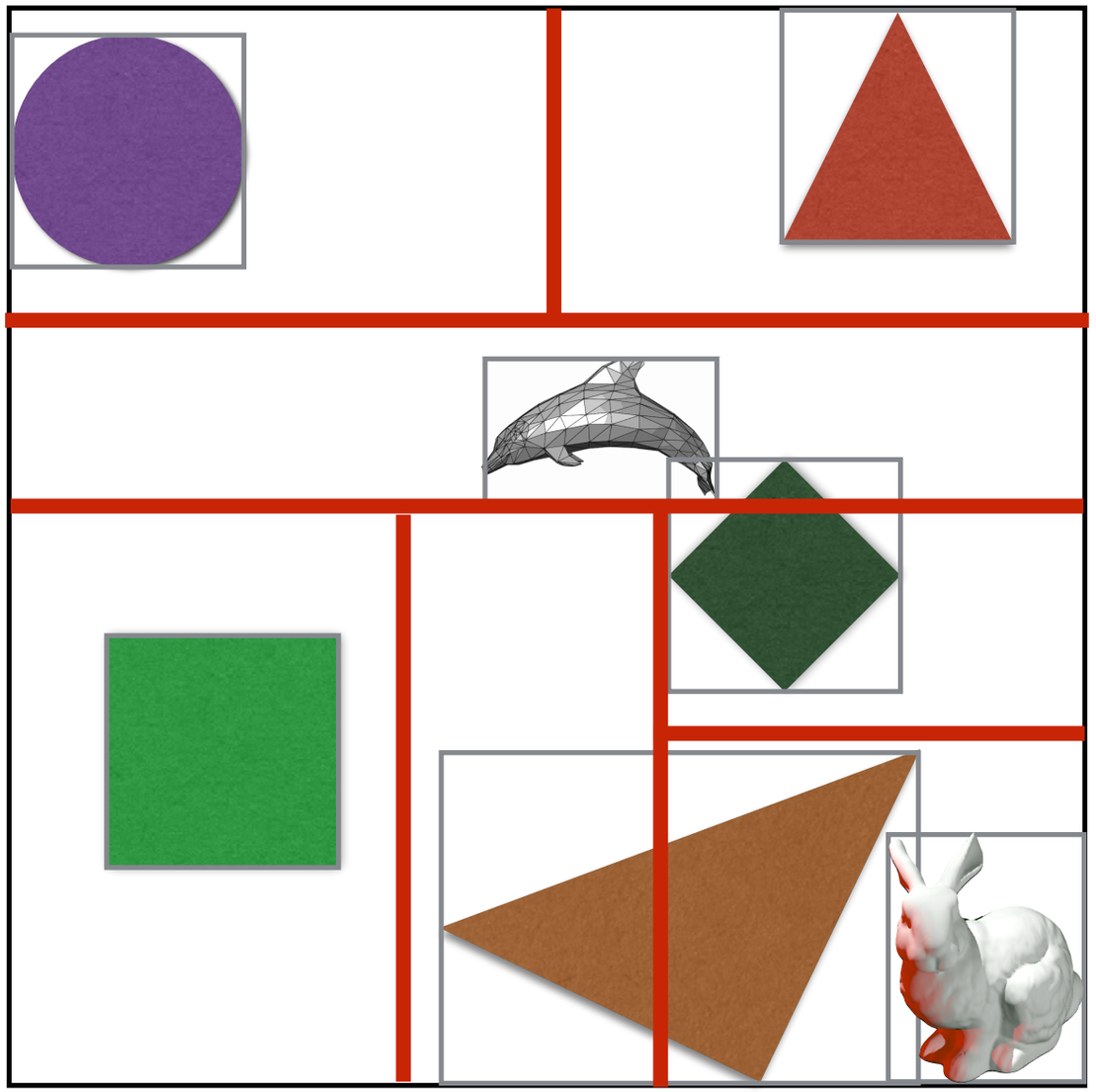
- The easy way
  - first divide along x axis, then y, then z, then x, etc.
  - always divide the space in half
- This will result in poor performance of the hit function
- Rather: optimise the constructed kd-tree by following some heuristics

Heuristics

1. Divide along the largest dimension
2. Divide such that both sides have roughly the same number of shapes
3. Avoid dividing along a plane which causes many overlaps (i.e. with many shapes on both sides)

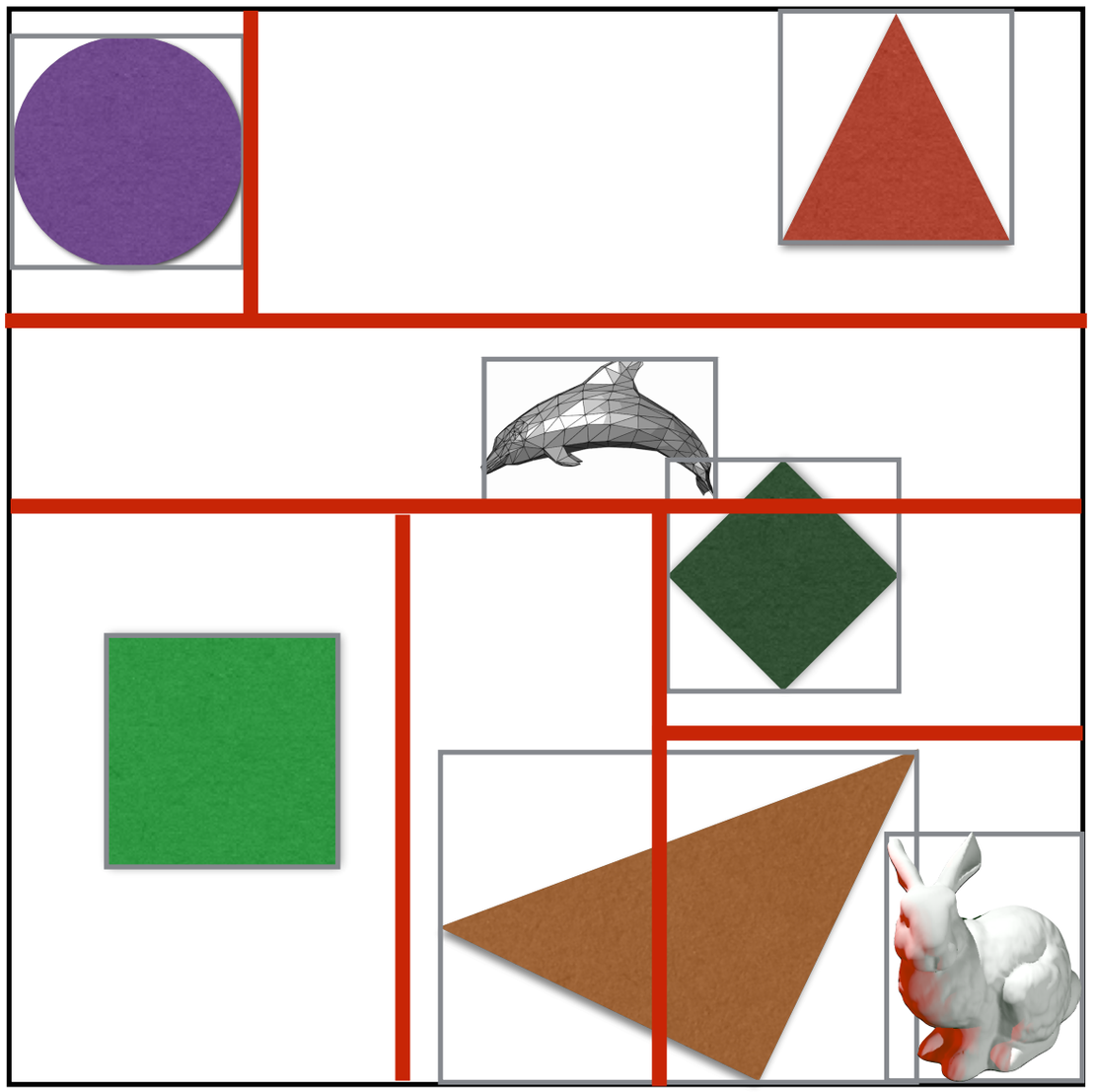
Heuristics

4. Cut off empty space as soon as possible



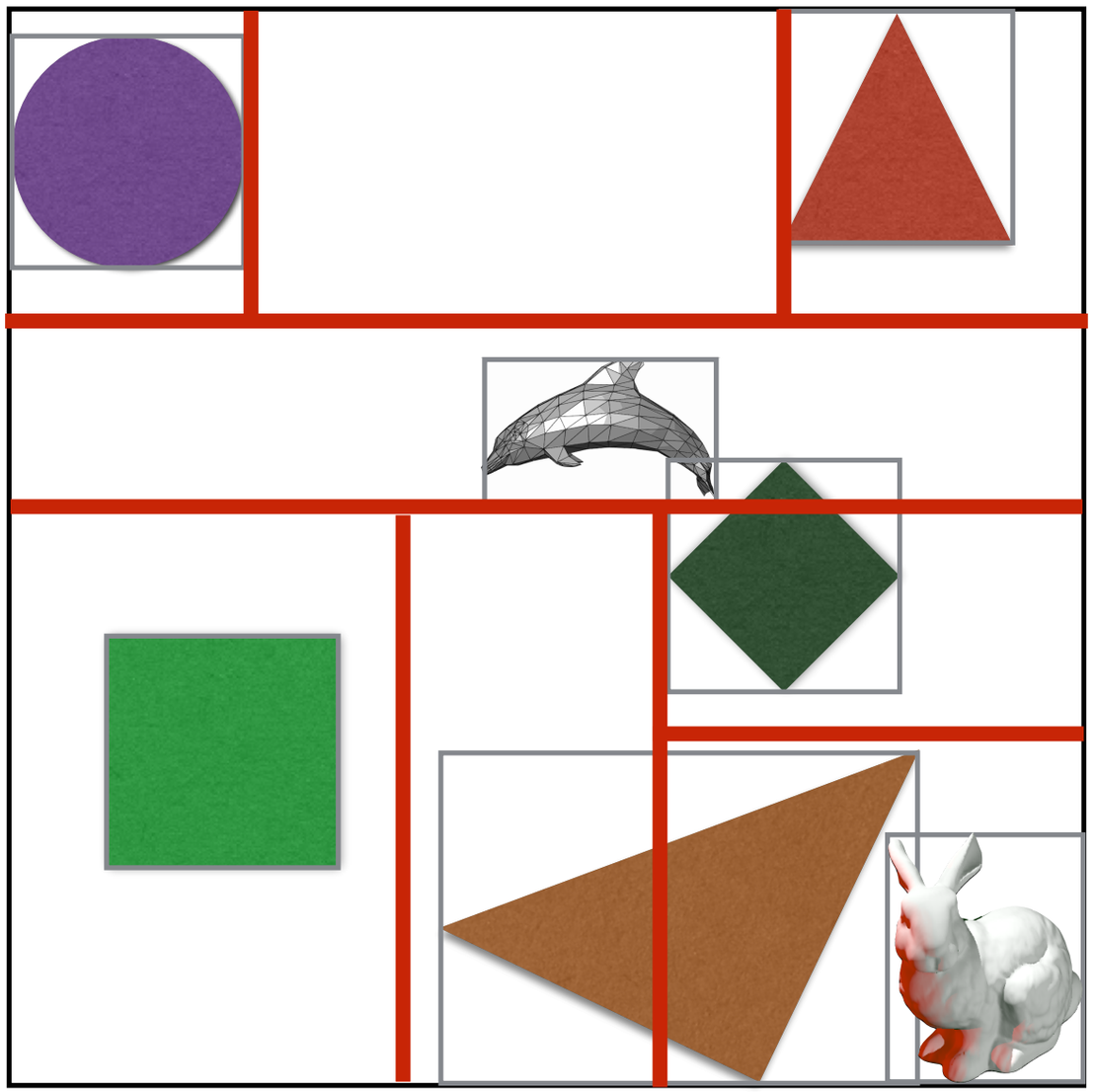
Heuristics

4. Cut off empty space as soon as possible



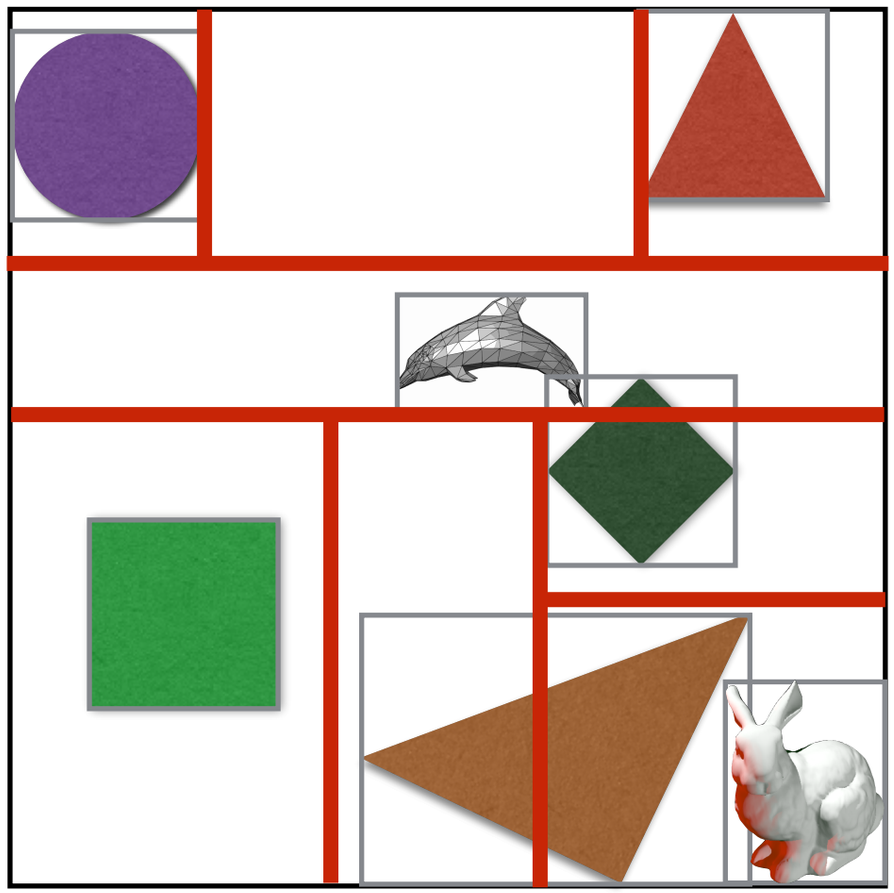
Heuristics

4. Cut off empty space as soon as possible



Heuristics

5. Don’t divide any further if 60% of shapes on one side are also on the other side



Meta-Heuristics

- One kd-tree per triangle mesh
  - same kd-tree is shared for every instance of the triangle mesh
- One kd-tree for the whole scene (i.e. contains all shapes that have a bounding box)
- keep the code simple, optimise later

kd-Tree Traversal

```
traverse(tree, ray) =
  if intersect(tree.bbox, ray) = Some (t, t’)
  then search(tree.root, ray, t, t’)
  else None

search(node, ray, t, t’) =
  if isLeaf node
  then search-leaf(node, ray, t’)
  else search-node(node, ray, t, t’)

search-leaf(leaf, ray, t’) =
  if closestHit(leaf.shapes, ray) = Some hit
        && hit.distance < t’
  then Some hit
  else None
```

kd-Tree Traversal

```
search-node(node, ray, t, t’)
  a = node.axis
  if ray.direction[a] = 0
  then if ray.origin[a] <= node.value
    then search(left, ray, t, t’)
    else search(right, ray, t, t’)
  else
    tHit = (node.value - ray.origin[a]) / ray.direction[a]
    (fst, snd) = order(ray.direction[a], node.left, node.right)
    if tHit <= t || tHit < 0 then
      search(snd, ray, t, t’)
    else if tHit >= t’ then
      search(fst, ray, t, t’)
    else
      if search(fst, ray, t, tHit) = Some hit
      then Some hit
      else search(snd, ray, tHit, t’)
```

kd-Tree Traversal

```
search-node(node, ray, t, t')
  a = node.axis
  if ray.direction[a] = 0
  then if ray.origin[a] <= node.value
    then search(left, ray, t, t')
    else search(right, ray, t, t')
  else
    tHit = (node.value - ray.origin[a]) / ray.direction[a]
    (fst, snd) = order(ray.direction[a], node.left, node.right)
    if tHit <= t || tHit < 0 then
      search(snd, ray, t, t')
    else if tHit >= t' then
      search(fst, ray, t, t')
    else
      if search(fst, ray, t, tHit) = Some hit
      then Some hit
      else search(snd, ray, tHit, t')
```

```
order(d, left, right) =
  if d > 0
  then (left, right)
  else (right, left)
```

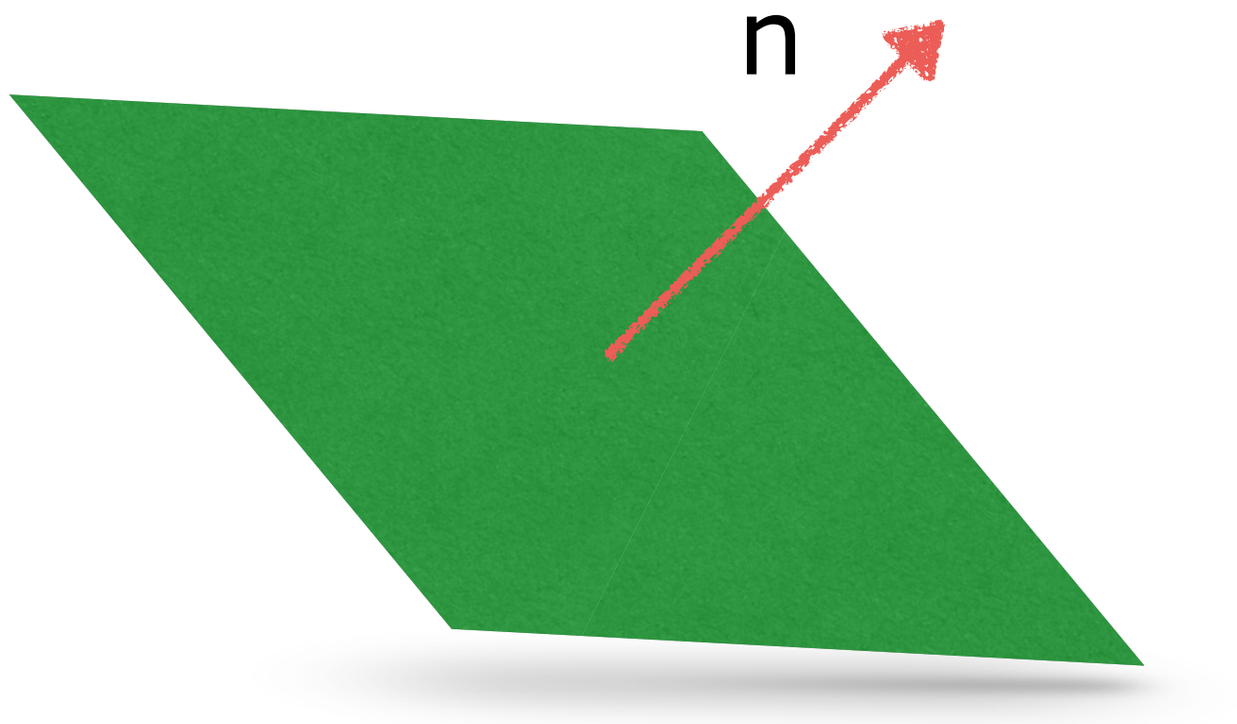
IT UNIVERSITY OF COPENHAGEN

Questions

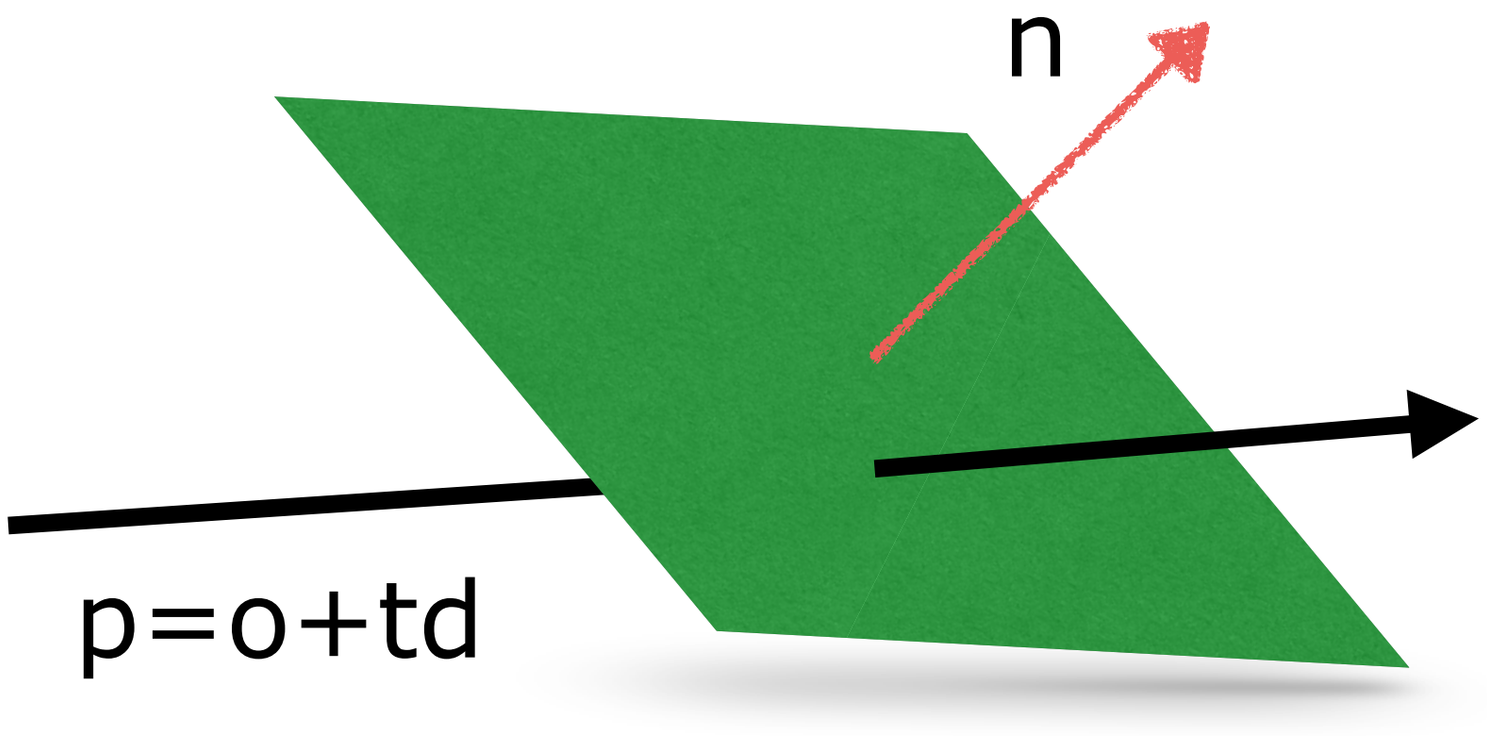
Miscellanea

- Negate normals if hit from other side
- Coloured lights
- Transformations on bounding boxes
- merge transformations: inverse matrices have to be multiplied in reverse order!
- gamma correction

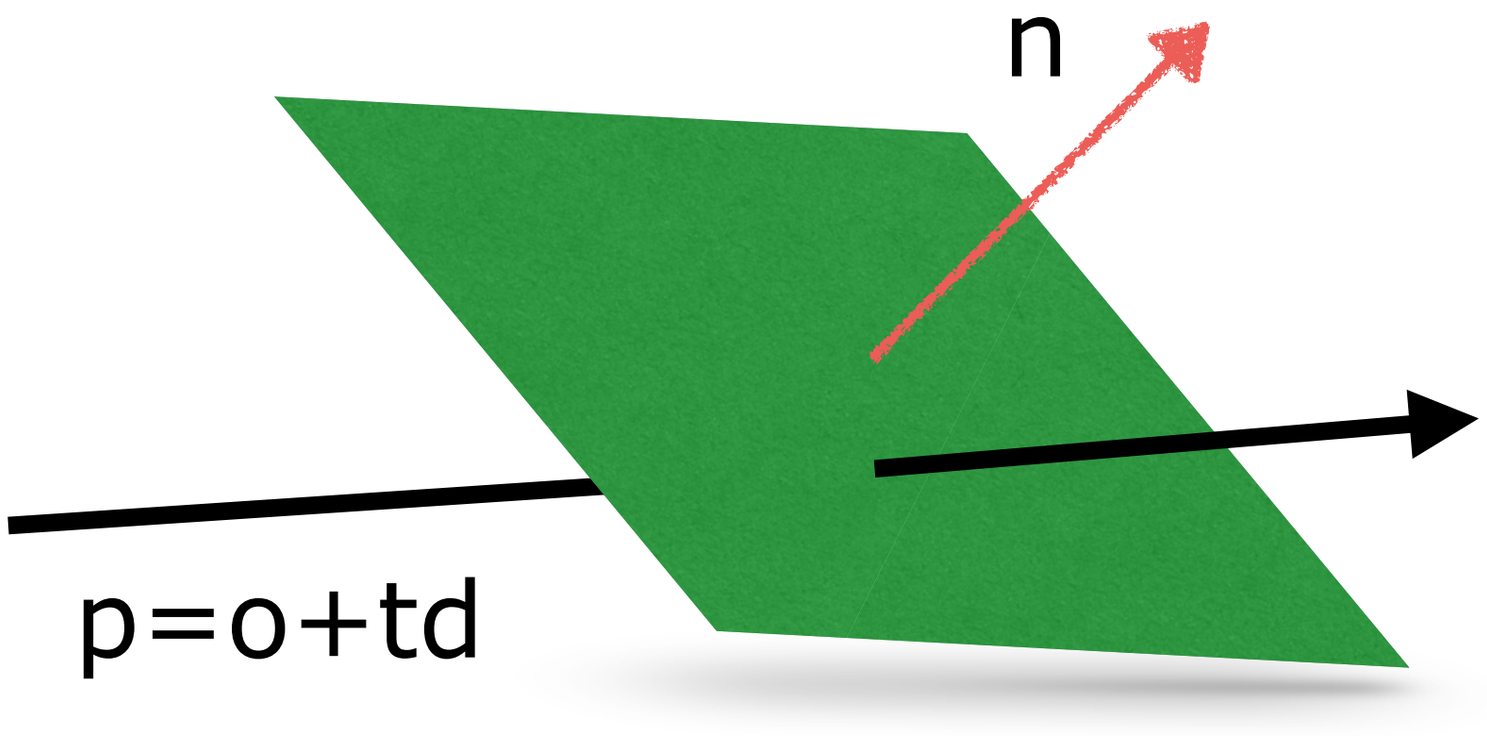
Negate Normal Vector



Negate Normal Vector

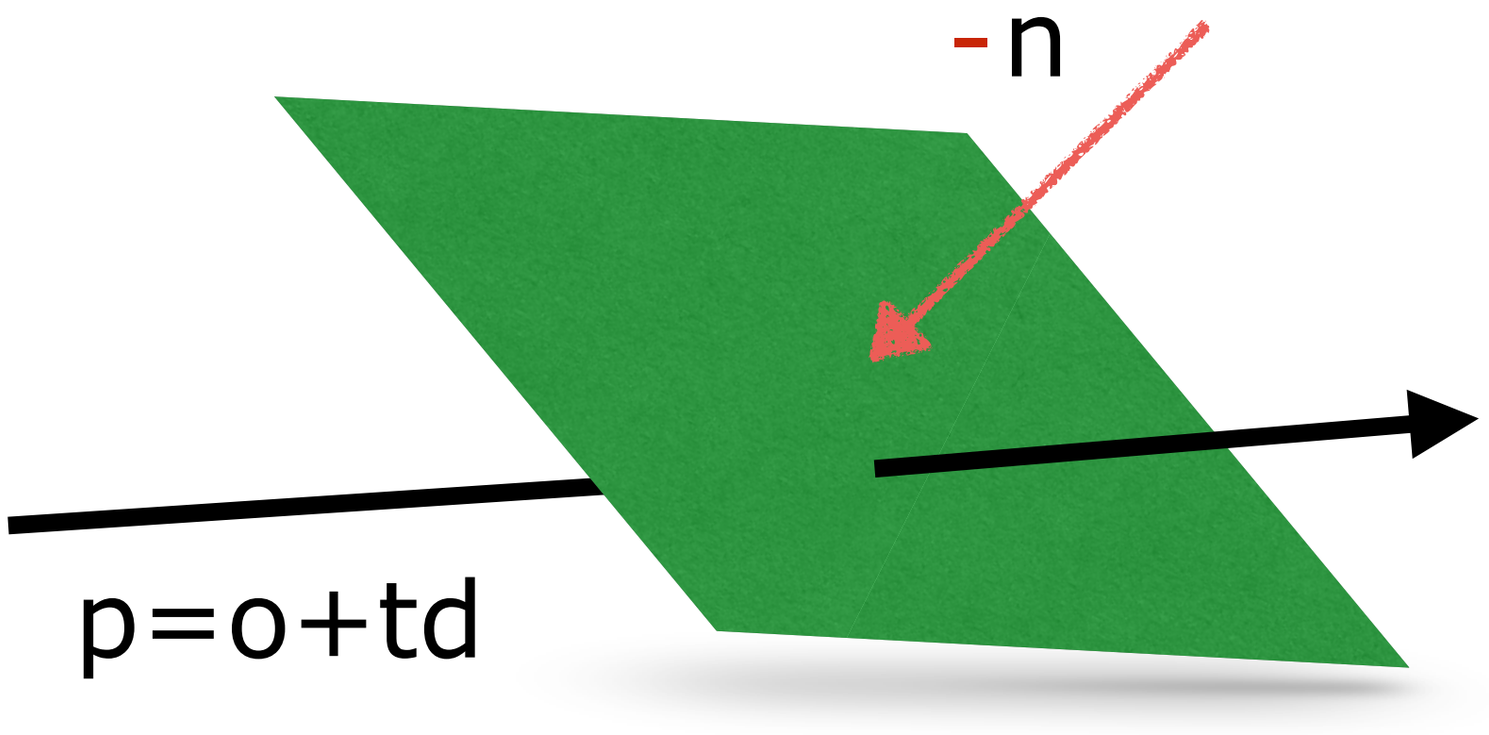


Negate Normal Vector



negate n if $d \cdot n > 0$

Negate Normal Vector



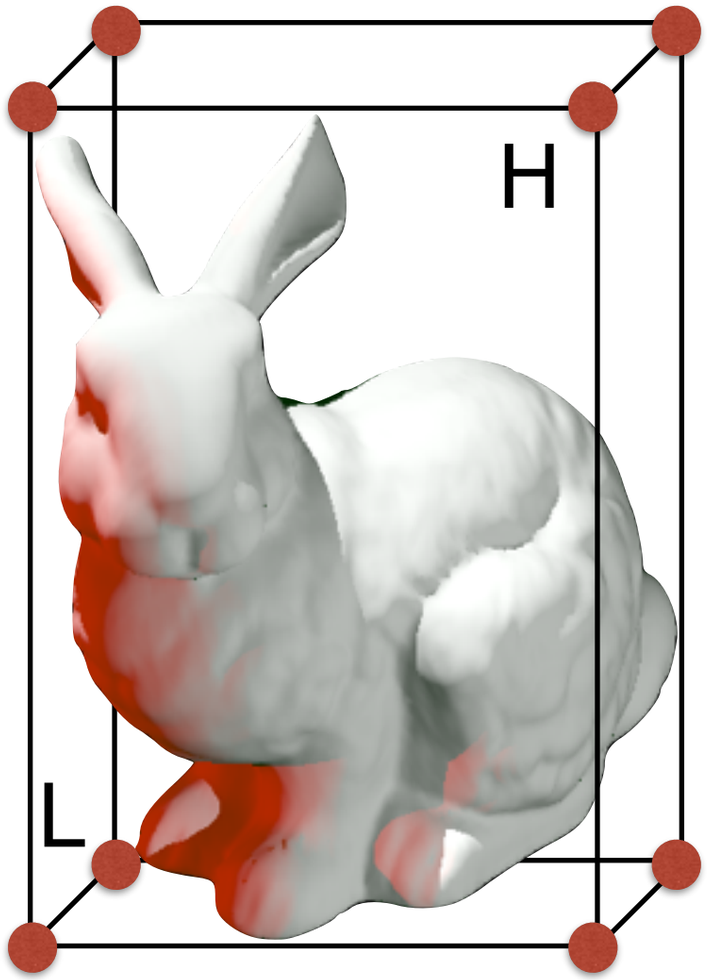
negate n if $d \cdot n > 0$

Coloured Lights

- light has colour (r,g,b) and intensity i
- calculate light intensity for each colour
- $i_r = i \cdot r$, $i_g = i \cdot g$, $i_b = i \cdot b$
- to light a material of color (r_m,g_m,b_m) calculate $(r_m \cdot i_r, g_m \cdot i_g, b_m \cdot i_b)$

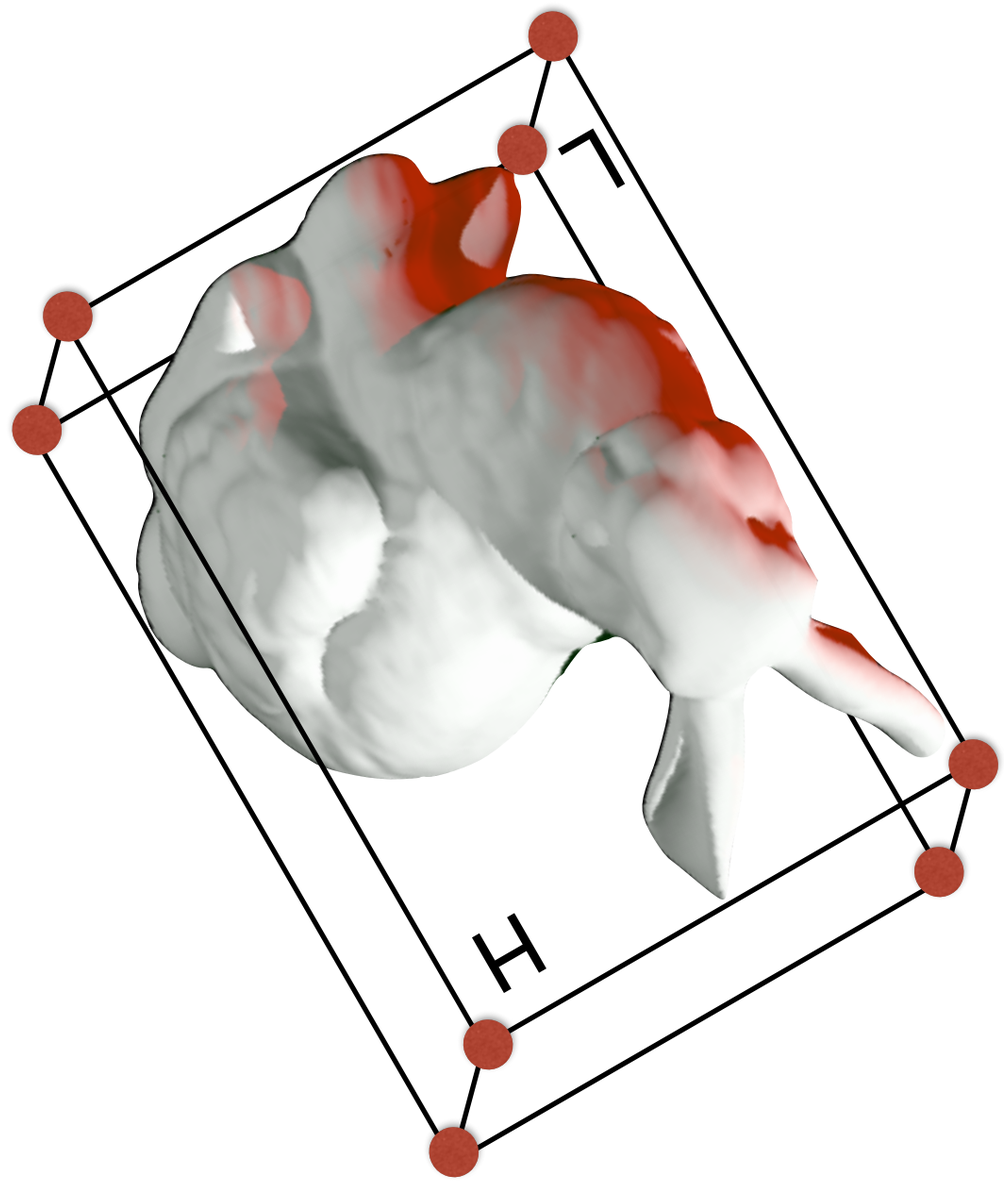
Rotate Bounding Boxes

- calculate all vertices of the bounding box
- transform all vertices according to the matrix (not the inverse!)
- calculate minimum and maximum coordinates of the transformed vertices
- This is only needed for constructing kd-trees



Rotate Bounding Boxes

- calculate all vertices of the bounding box
- transform all vertices according to the matrix (not the inverse!)
- calculate minimum and maximum coordinates of the transformed vertices
- This is only needed for constructing kd-trees



Combine Transformations

- If we have matrices $M_1, M_2, \ldots, M_n$
- $M_n \cdot M_{n-1} \cdot \ldots \cdot M_1$ is the matrix for the combined transformation (applied left-to-right)
- $M_1^{-1} \cdot M_2^{-1} \cdot \ldots \cdot M_n^{-1}$ is the inverse matrix for the combined transformation (applied left-to-right)

Gamma Correction

- ray tracer works with floating point colour values (between 0 and 1)
- obtained from integer colour values by dividing by 255
- additionally the resulting value is raised to the power of 2 ($\gamma = 2$)
- before translating back to integer colour values: raise floating point value to the power of 1/2 (i.e. $1/\gamma$)