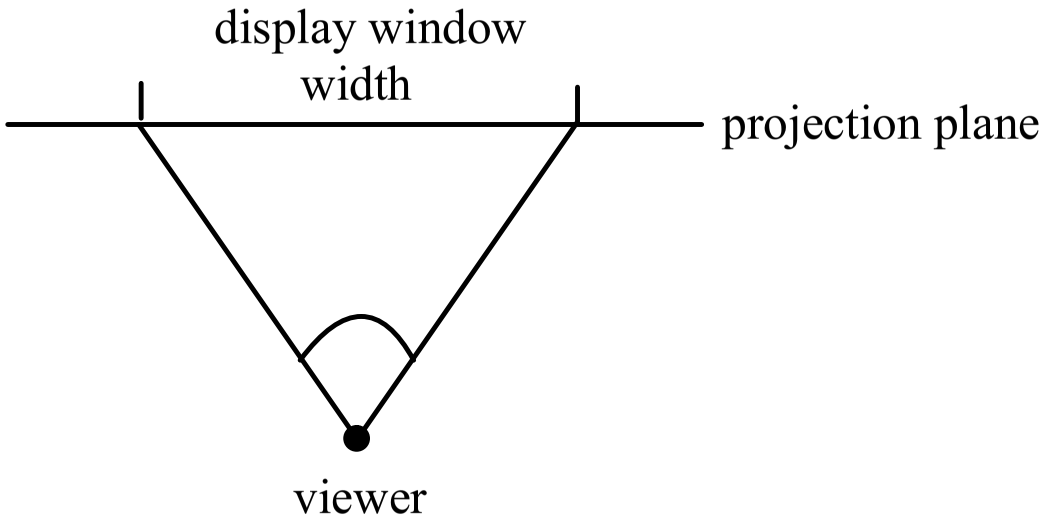
The viewer in the virtual world

Specify:

- The coordinates of the point where the viewer stands
- The projection plane
- The type of projection (perspective/parallel)
- The field of view, given by an angle determining the width that the the window on the computer screen will have on the projection plane.

Angle of view



In Java 3D:

```
View v =
  simpUniv.getViewer().getView();

v.setFieldOfView(angle);
```

The viewer in the virtual world

In Java 3D, the viewer is characterised by an instance of the class `PhysicalBody`.

The distance of the projection plane to the viewer can be defined by

```
v.getPhysicalBody().
setNominalEyeOffsetFromNominalScreen(
distance);
```

The viewer in the virtual world

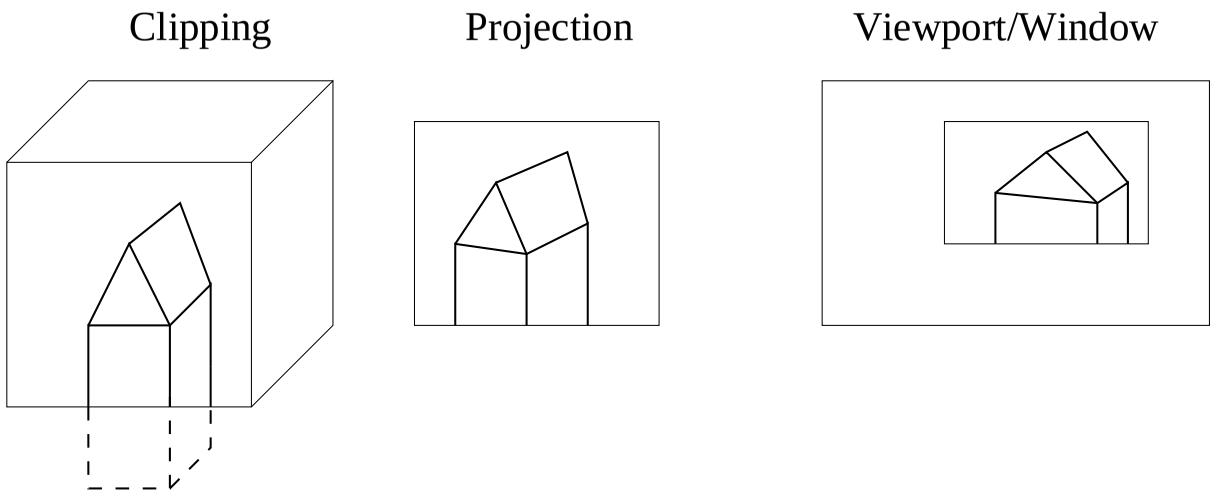
Changing the position of the viewer by a transformation `vt`:

```
Transform3D vt = new Transform3D();

vt.set(...);

simpUniv.getViewPlatform.setTransform(vt);
```

3D clipping



Clipping refers to determining which objects or which parts of the objects are within the field of view.

Clipping does not include visibility considerations concerning objects that are hidden from view by other object.

3D clipping

Theoretically, clipping in means the restriction of the scene to

- a pyramid of infinite height in case of perspective projection and
- to a box with infinite extension in one direction in case of parallel projection.

The distance a person can see is almost unlimited. But . . .

3D clipping

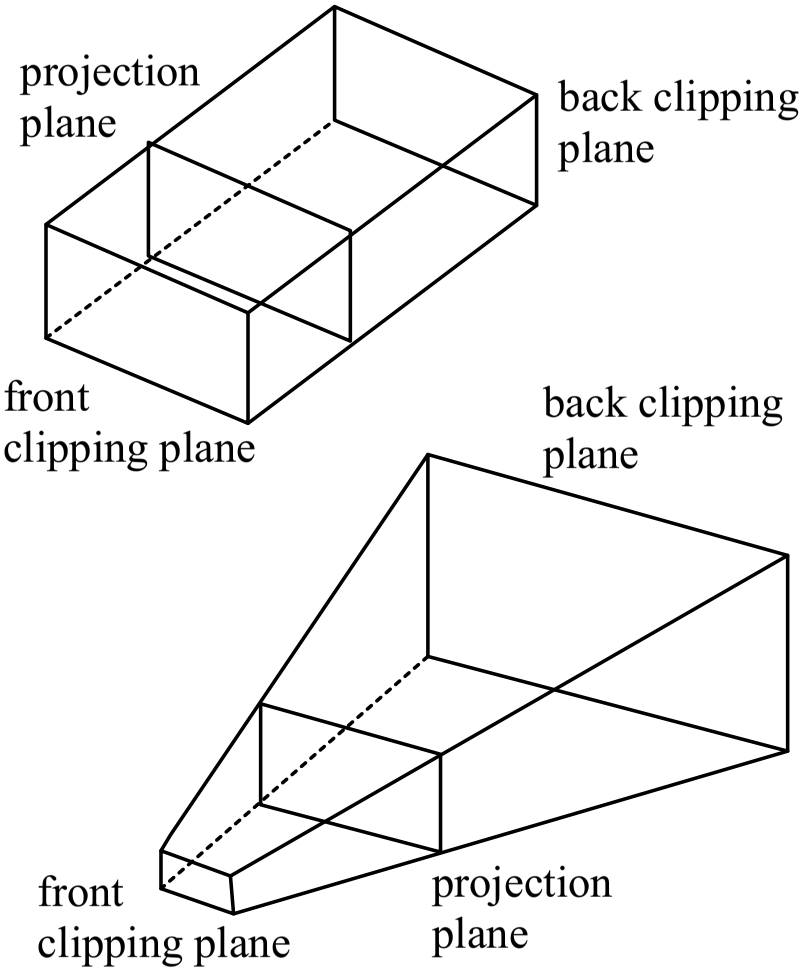
. . . it is impossible to focus the eyes to birds in the sky and a finger close to the eyes at the same time.

The eyes can adjust to a specific distance.

Only objects approximately in this distance are in focus.

This fact is modelled in computer graphics by clipping planes, the **front** and the **back clipping plane**.

3D clipping



3D clipping

Parallel projection: The clipping region is a box.

Perspective projection: The clipping region is a frustum of pyramid.

Usually, the projection plane lies between the front and the back clipping plane.

- projection plane $\widehat{=}$ computer screen
- front clipping plane $\widehat{=}$ minimum distance in focus
- back clipping plane $\widehat{=}$ maximum distance in focus

3D clipping

Clipping for parallel projection onto the x/y-plane:

Clipping region: Box, determined by the two vertices $(x_{\text{min}}, y_{\text{min}}, z_{\text{min}})$ and $(x_{\text{max}}, y_{\text{max}}, z_{\text{max}})$.

Checking whether a point (p_x, p_y, p_z) lies within the clipping volume:

$$x_{\text{min}} \leq p_x \leq x_{\text{max}} \quad \text{and}$$
$$y_{\text{min}} \leq p_y \leq y_{\text{max}} \quad \text{and}$$
$$z_{\text{min}} \leq p_z \leq z_{\text{max}}$$

3D clipping

In Java 3D:

```
v.setBackClipDistance(bcDist);

v.setFrontClipDistance(fcDist);
```

The object `v` is the `View` belonging to the `SimpleUniverse`.

(Clipping and angle of view, see `ClippingPlanes.java`)

Visible surface determination

For displaying objects from a virtual 3D world, apart from clipping and projection, it is necessary to determine whether objects within the clipping volume are visible or hidden from view by other objects.

The problem of determining which objects are visible and which ones are not is referred to as **hidden line** and **hidden surface elimination** or **visible line** and **visible surface determination**.

Visible surface determination

Simple algorithm:

```
for (each pixel in the window)
{
  Determine the object with the smallest
  distance to the viewer which would be
  projected to the pixel.

  Assign the corresponding colour to
  the pixel.
}
```

This technique is called **image-precision algorithm**.

Visible surface determination

An image-precision algorithm has a complexity of $n \cdot p$ for an image with p pixels and n objects.

Alternative approach: **Object-precision algorithms**:

```
for (each object in the clipping volume)
{
  Determine the parts of the object which
  are not hidden from view by other
  objects.

  Draw the visible parts of the object
  in the corresponding colour.
}
```

Visible surface determination

An object-precision algorithm requires n^2 steps.

In most cases $n \ll p$ holds, implying $n^2 \ll np$.

However, the single steps of object-precision algorithms are much more complex than those of image-precision algorithms.

One advantage of object-precision algorithms is that they work independent of the resolution.

Back-face culling

Back-face culling: Remove all polygons that do not face the viewer. They are neither visible nor can they hide other polygons from view.

Polygons not facing the viewer can be identified by their associated normal vectors.

The normal vectors are chosen in such a way that they point to the outside of the surface.

Back-face culling

A polygon is a back-face (and can be ignored for rendering).

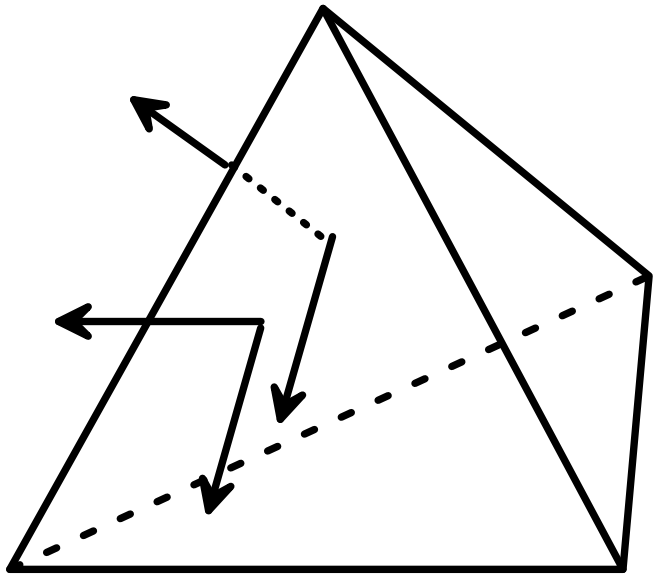
$\Longleftrightarrow$ The angle between the normal vector and the z-axis (the direction of projection) is larger than $90°$.

$\Longleftrightarrow$ The dot product of the normal vector and the z-unit vector is negative.

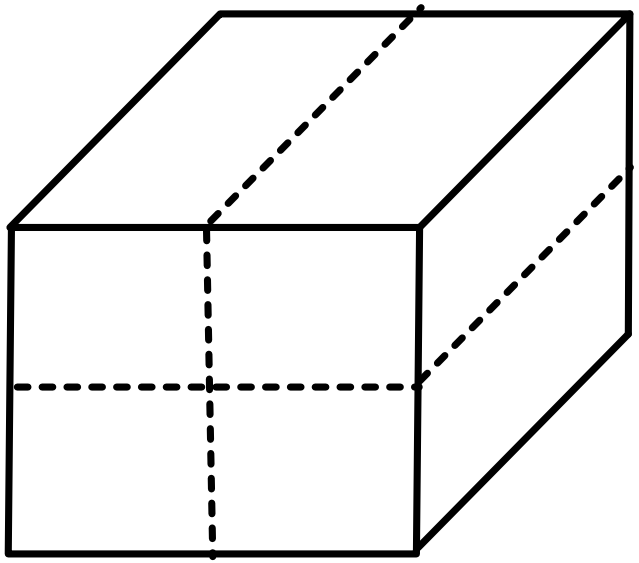
$\Longleftrightarrow$ The z-coordinate of the normal vector is negative.

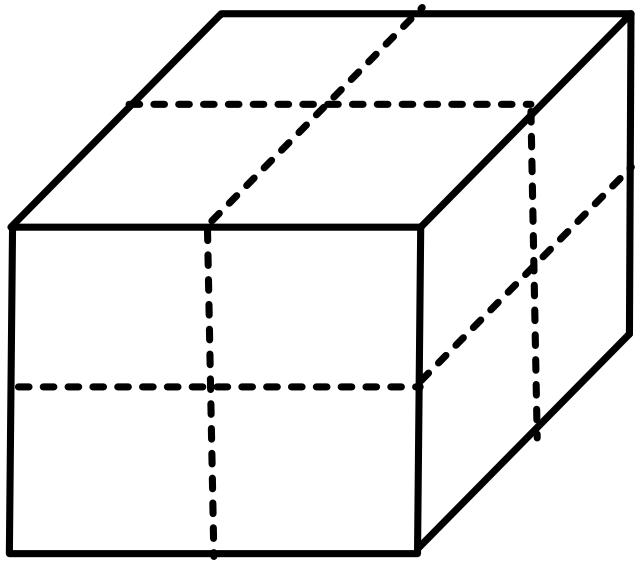
(for a parallel projection to a plane parallel to the x/y-plane)

Back-face culling



Spatial partitioning





Spatial partitioning

Partitioning of the clipping volume into smaller boxes.

Objects in boxes that are next to each other, but not in front or behind each other, cannot hide each other from view.

Objects in a box in front cannot be hidden from objects in boxes in the back.

Ideal case: Reduction of the computational costs from n^2 for n objects to $k \cdot \left(\frac{n}{k}\right)^2 = \frac{n^2}{k}$ for k boxes.

Not valid for larger k: Objects lie in more than one small box.

Recursive subdivision algorithms

Recursive subdivision of the clipping volume into smaller boxes, until at most one object is within each box.

The resolution limits the maximum level of recursion.

Area subdivision algorithms: Partitioning the projection plane.

Octree algorithms: Partitioning the clipping volume as in the case of object modelling with octrees.

The z-buffer algorithm

Based on a frame buffer for the colours of the pixels in the image and a z- or depth-buffer in which a z-value is entered for each pixel.

The frame buffer is initialised with the background image or background colour.

The objects are projected in an arbitrary order and the projections are stored in the frame buffer.

The z-buffer algorithm

Before a pixel of a projected object is entered into the frame buffer, its z-value, i.e., its distance to the projection plane, is compared to the value entered for the corresponding pixel so far in the z-buffer.
If the value in the z-buffer is larger than the z-value of the point of the object that led to the projected pixel, then the new pixel colour is entered into the frame buffer and the z-value is also updated.

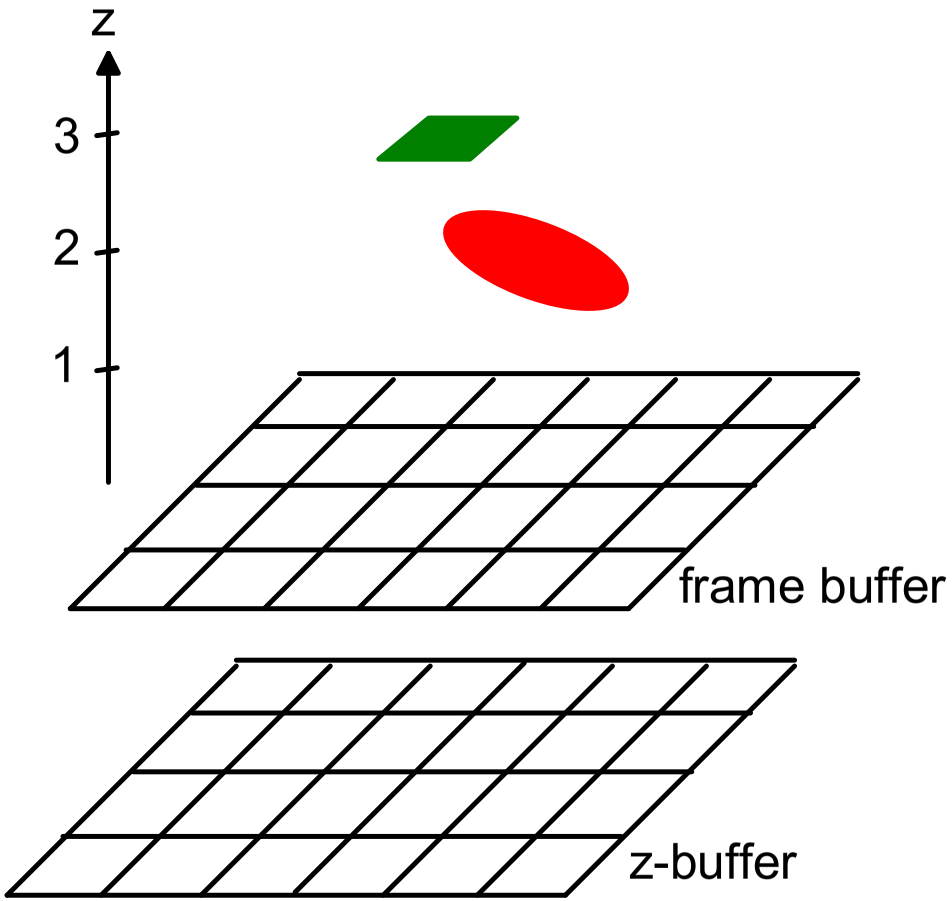
The order in which the objects are projected does not matter.

Additional objects can be entered easily.

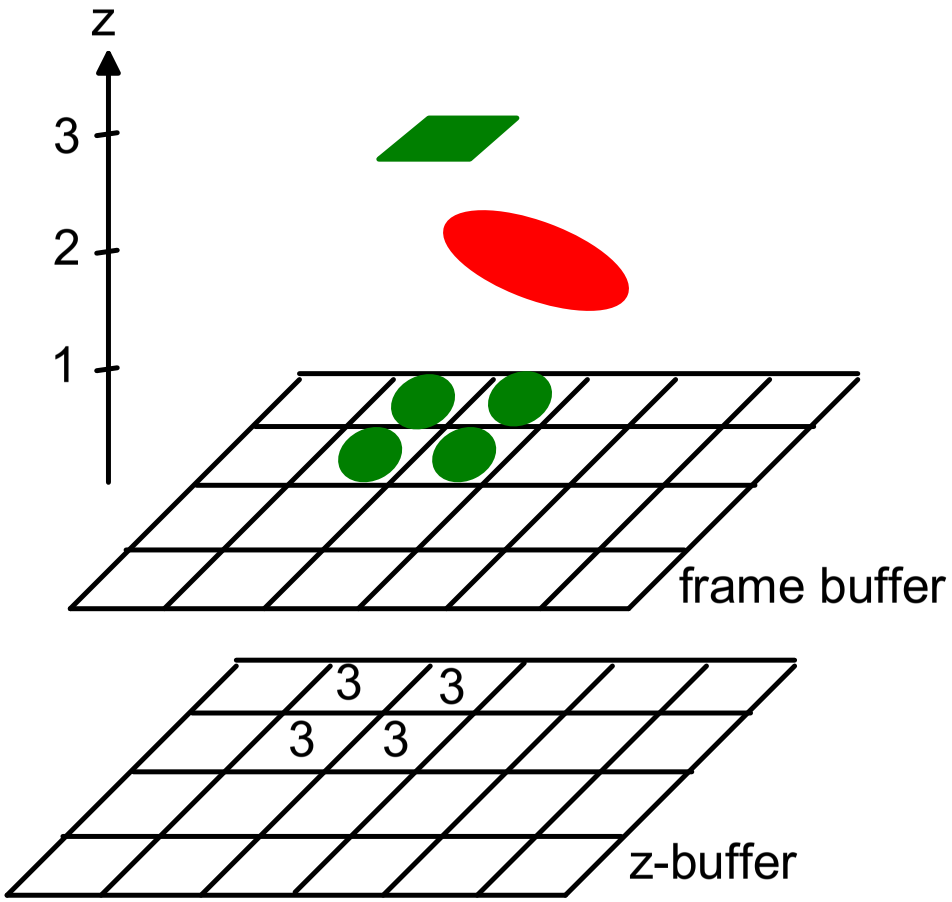
The z-buffer algorithm

For image sequences with fixed background, only the moving objects have to be entered again into the buffers. All static objects can be entered once into the buffers as a “background”.

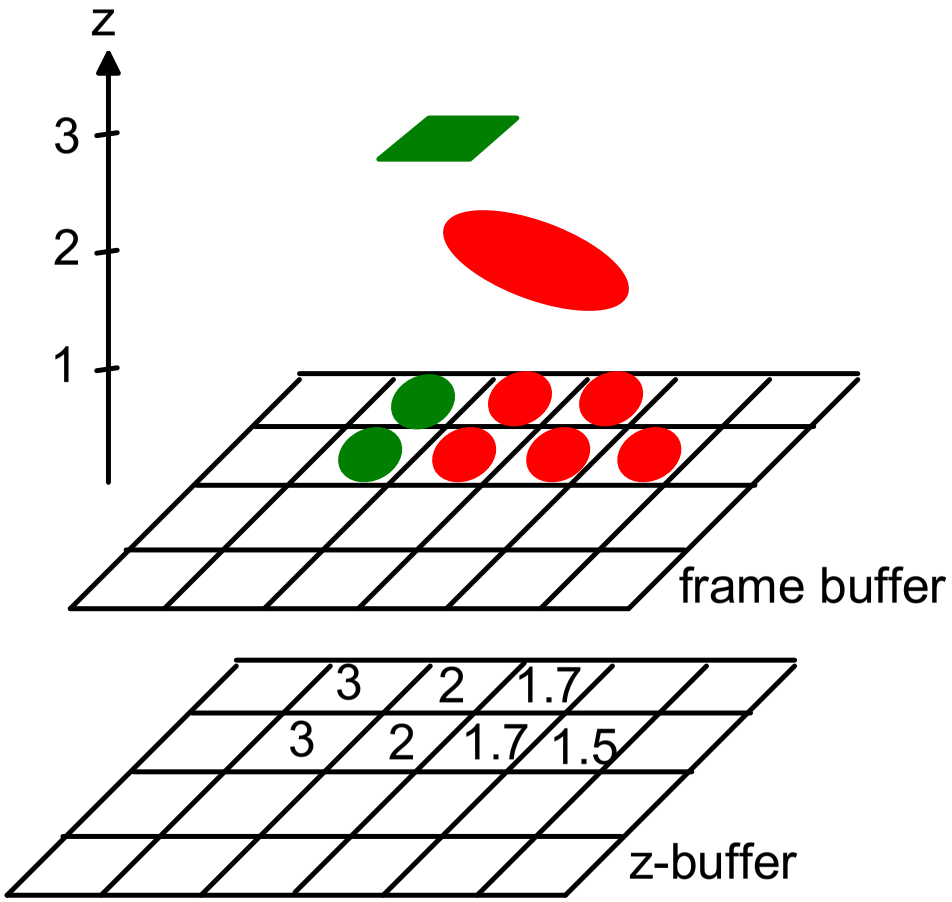
The z-buffer algorithm



The z-buffer algorithm



The z-buffer algorithm



Scan line technique

Entering a polygon using a scan line technique:

Equation for the plane induced by the polygon:

$$A \cdot x + B \cdot y + C \cdot z + D = 0$$

Computing the z-value along the scan line:

$$z_{\text{new}} = z_{\text{old}} + \Delta z$$

Let z_{old} be the z-coordinate of the projected polygon at pixel (x, y).

Scan line technique

Determine the z-coordinate for the following pixel $(x+1, y)$:

$$
\begin{aligned}
0 &= A \cdot (x+1) + B \cdot y + C \cdot z_{\text{new}} + D \\
&= A \cdot (x+1) + B \cdot y + C \cdot (z_{\text{old}} + \Delta z) + D \\
&= \underbrace{A \cdot x + B \cdot y + C \cdot z_{\text{old}} + D}_{=0} + A + C \cdot \Delta z \\
&= A + C \cdot \Delta z
\end{aligned}
$$

Therefore: $\Delta z = -\frac{A}{C}$

Scan line technique

Scan line techniques carry out computations along pixel rows (or pixel columns).

Pixel coordinates: (u, v)

Use three tables:

Edge table: Contains all nonhorizontal edges:

v_{min}	$u(v_{\text{min}})$	v_{max}	Δu	Polygon no.

Scan line technique

- v_{min}: smallest v-value of the edge
- $u(v_{\text{min}})$: u-value corresponding to the v_{min}-value of the edge
- v_{max}: largest v-value of the edge
- Δu: Increment (slope) of the edge
- Polygon no.: List of polygons to which the edge belongs

The edges are sorted in increasing order with respect to their v_{min}-values. For identical v_{min}-values, edges with a smaller $u(v_{\text{min}})$-value come first.

Scan line technique

Polygon table: Contains all polygons:

Polygon no.	A	B	C	D	Colour	In-flag

- Polygon no.: Identifier for the polygon
- A, B, C, D define the plane associated with the polygon: $Ax + By + Cz + D = 0$
- Colour: Colour value or colour information for the polygon
- In-flag: Indicates, whether the actual position on the scan line lies within or outside the polygon.

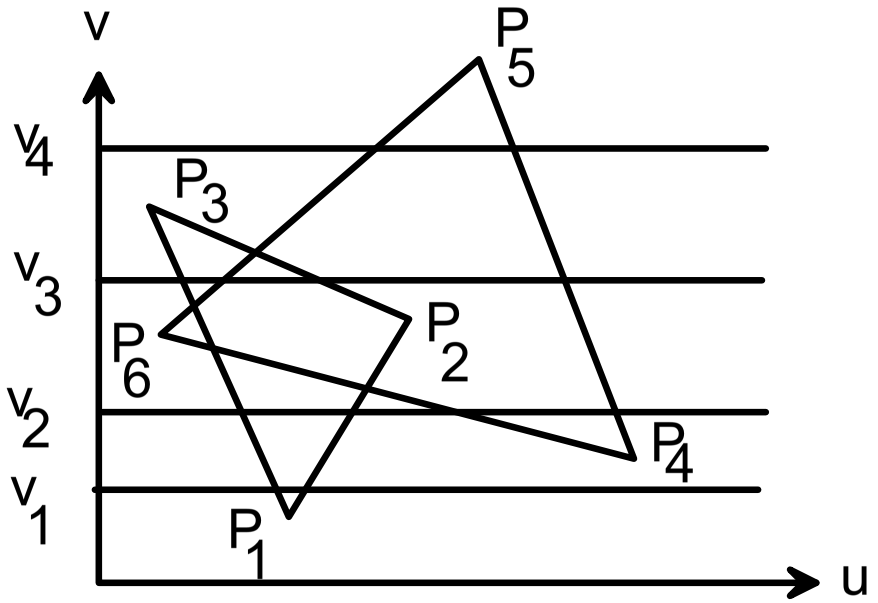
Scan line technique

Active edges table: List of all edges intersecting the considered scan line. Sorted in increasing order by the u-components of the intersection points.

The number of rows of the table of active edges can be different for each scan line.

The number of rows and the entries of the other two tables remain constant except for the entries in the column with the In-flag.

Scan line technique



Active edges for the scan lines v_1, v_2, v_3, v_4.

$$v_1 : P_3P_1,\ P_1P_2$$
$$v_2 : P_3P_1,\ P_1P_2,\ P_6P_4,\ P_5P_4$$
$$v_3 : P_3P_1,\ P_6P_5,\ P_3P_2,\ P_5P_4$$
$$v_4 : P_6P_5,\ P_5P_4$$

Scan line technique

For a scan line apply the following:

```
Update the list of active edges;
Set all In-flags to 0;
for (all intersection points)
    (see active edge list)
{
  Update the In-flags;
  Determine the visible polygon;
  Choose the pixel colour according
    to the entry in the polygon table;

}
```

Scan line technique

Determination of the visible polygon:

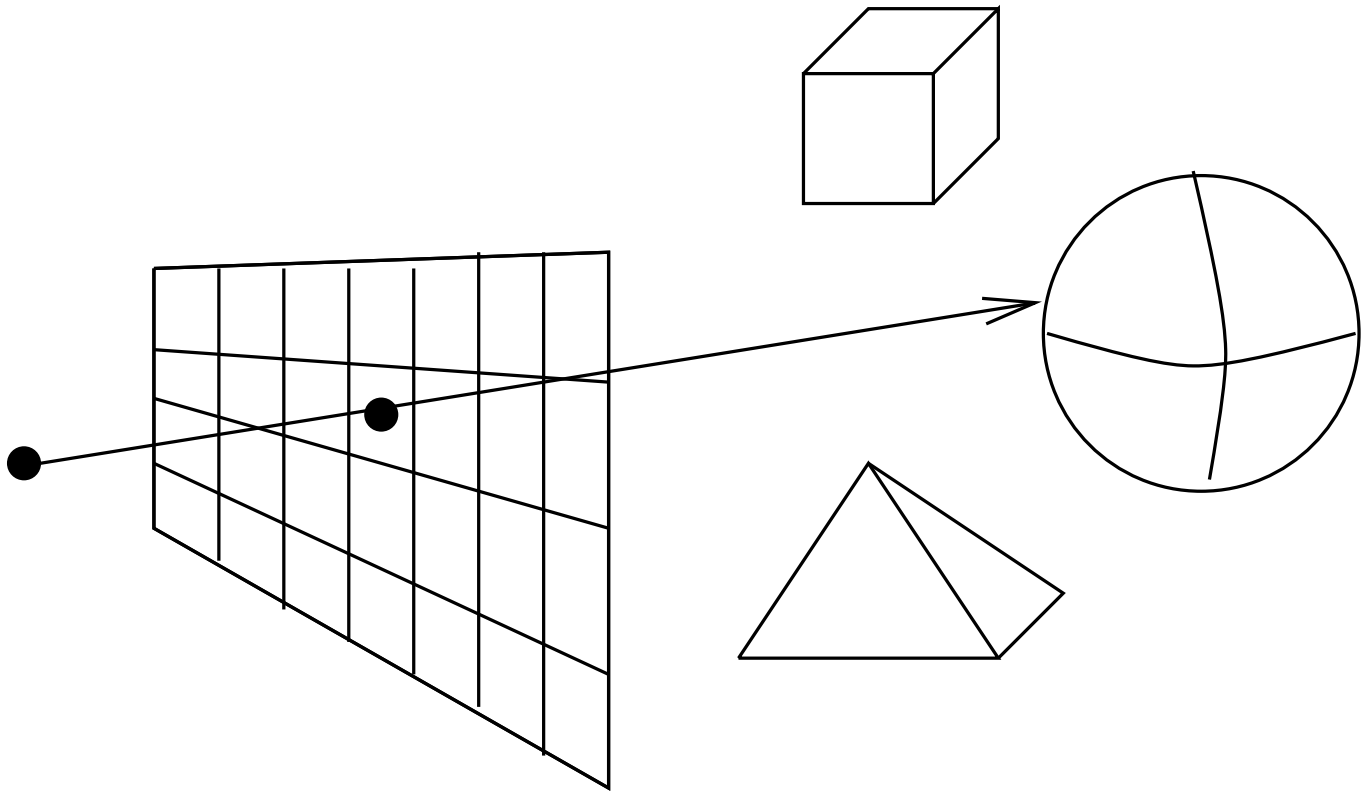
For each active polygon (In-flag = 1) derive the corresponding z-value from the equation for the plane

$$A \cdot u + B \cdot v + C \cdot z + D \; = \; 0.$$

The polygon with the smallest z-value is visible at the corresponding pixel.

Ray casting

For each pixel in the clipping rectangles of the projection plane a ray parallel to the direction of projection is cast.
The first object that the ray meets determines the colour of the pixel.



Ray casting

Parametrisation of the ray from (x_0, y_0, z_0) (i.e. centre of projection) to the point $P = (x_1, y_1, z_1)$ (i.e. point/pixel on the projection plane):

$$x = x_0 + t\Delta x, \quad y = y_0 + t\Delta y, \quad z = z_0 + t\Delta z$$

where

$$\Delta x = x_1 - x_0, \quad \Delta y = y_1 - y_0, \quad \Delta z = z_1 - z_0$$

$t > 1 \quad \Leftrightarrow \quad P$ lies behind the projection plane.

$0 < t < 1 \quad \Leftrightarrow \quad P$ lies between the centre of projection and the projection plane.

$t < 0 \quad \Leftrightarrow \quad P$ lies in front of the centre of projection.

Ray casting

Intersection with a polygon:

1. Compute the intersection point of the ray with plane induced by the polygon.
2. Check, whether the intersection point lies inside the polygon.

Equation for the plane: $$Ax + By + Cz + D = 0$$

Insert the ray: $$t = -\frac{Ax_0 + By_0 + Cz_0 + D}{A\Delta x + B\Delta y + C\Delta z}$$

Ray casting

If the denominator is 0, the ray is parallel to the plane.

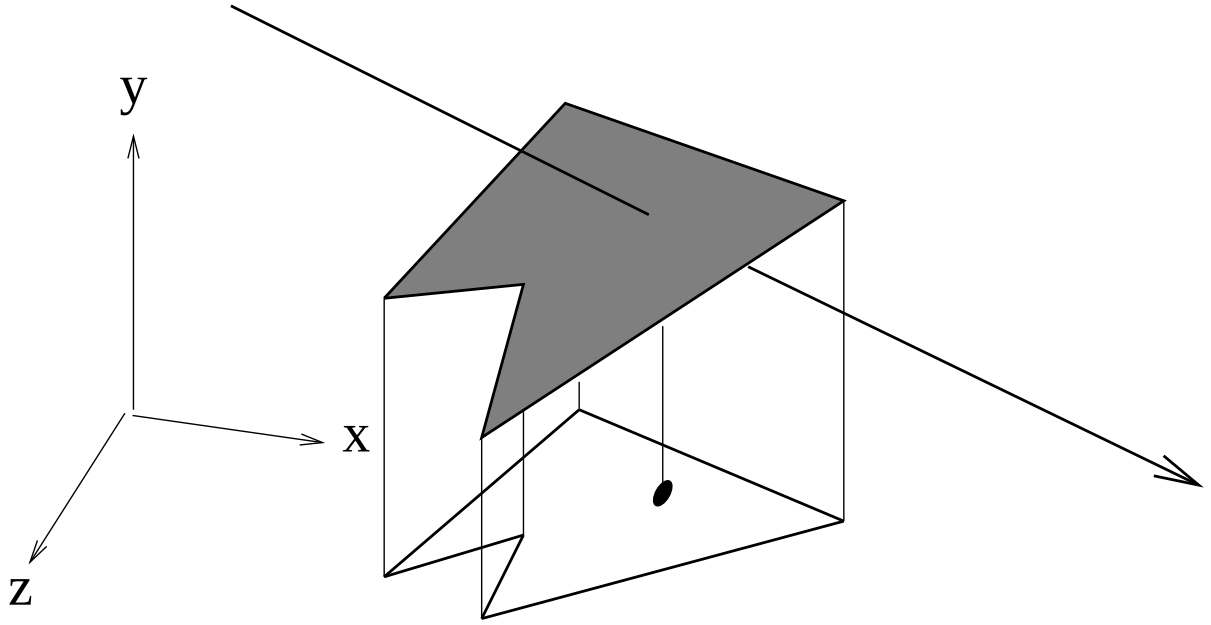
Otherwise, project the polygon and the intersection point to one of the planes defined by the axes of the coordinate system, setting one of the coordinates to zero.

To avoid problems with roundoff errors, the plane that should be chosen for projection should be the one that is most parallel to the plane of the polygon.

Ray casting

The projection plane is chosen orthogonal to the component which has the greatest absolute value in (A, B, C).

After the projection, the odd parity rule is applied to decide whether the intersection point lies within the projected polygon.



Ray casting

Coherence should be taken into account for ray casting in order to reduce the computational complexity.

Coherence refers to exploiting considerations like the following ones.

- Neighbouring pixels usually obtain their colour from the same polygon.
- Once a ray intersects a polygon, it is not necessary to calculate intersections with polygons which are farther away.

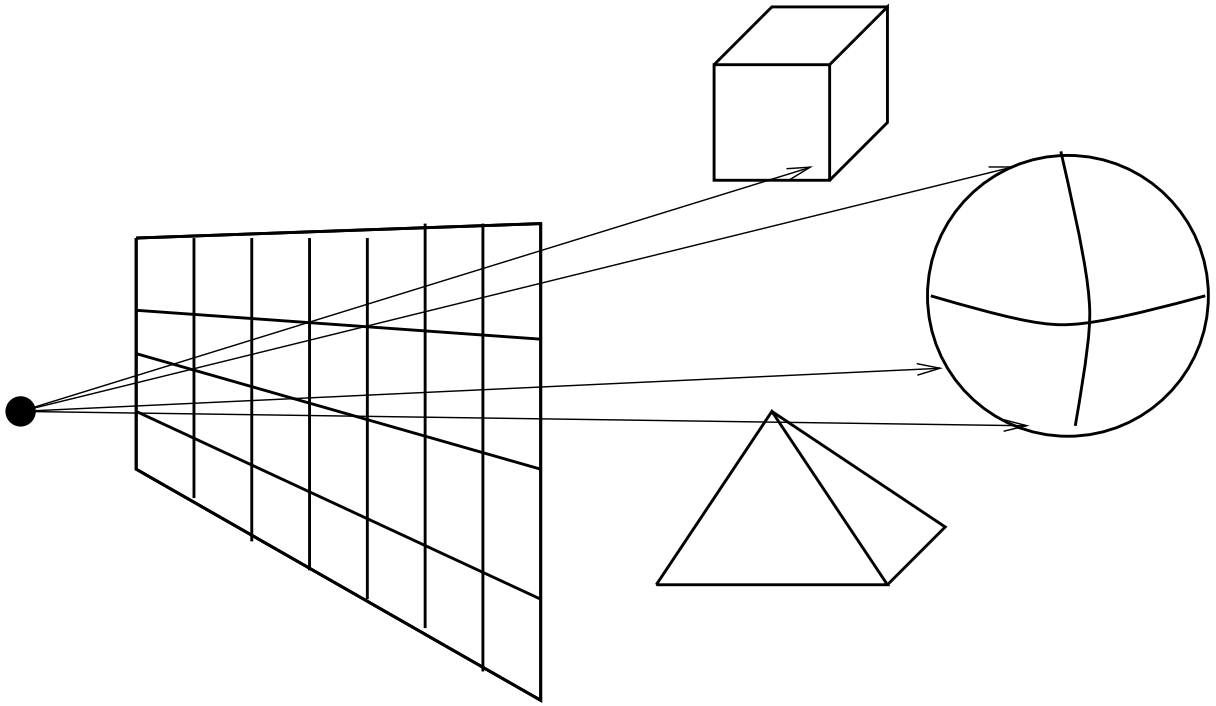
Ray-Casting

Without exploiting coherence, $1000 \cdot 1000 \cdot 100$, i.e., 100 million intersection tests would have to be carried out for a resolution of 1000×1000 pixels and 100 objects in the scene.

Ray casting can lead to aliasing effects when the back clipping plane is very far away.

Apply **supersampling** to avoid such aliasing effects: For one pixel more than one ray is cast. The colour of the pixel is calculated as the mean or weighted mean of the corresponding colours obtained from the objects that the rays meet.

Supersampling



Additional computational effort for $m \cdot n$ pixels:

$$(m+1) \cdot (n+1) - m \cdot n \;=\; m + n + 1$$

Priority algorithms

priority algorithms try to find a suitable order in which objects can be projected so that no conflicts occur during the projection.

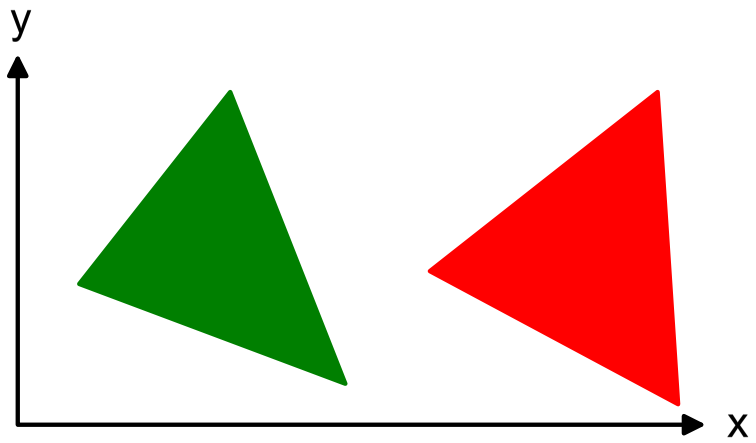
If there is no overlap of the z-coordinates of the two polygons, then the one with larger z-coordinates, the more distant one, is projected first.

In this way, closer objects will overwrite objects in the background.

Priority algorithms

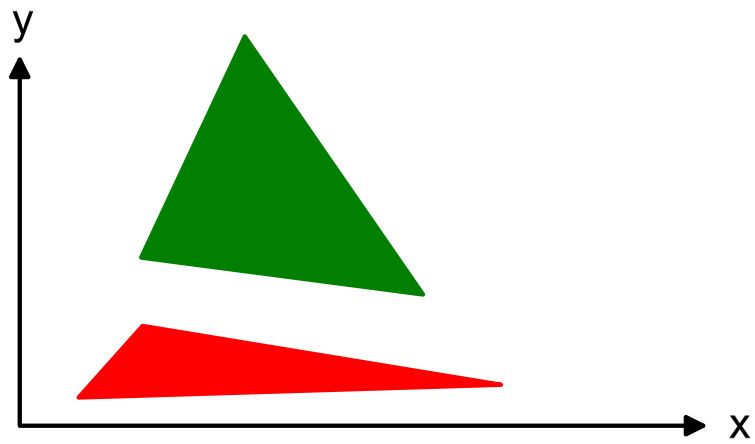
If the z-coordinates overlap, further tests have to be carried out.

1. Is there no overlap of the x-coordinates?



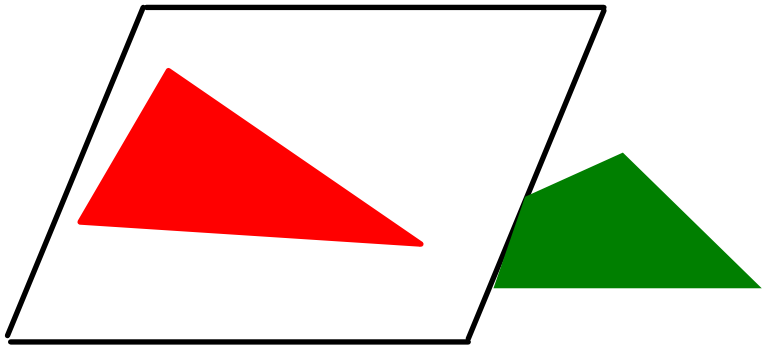
Priority algorithms

2. Is there no overlap of the y-coordinates?

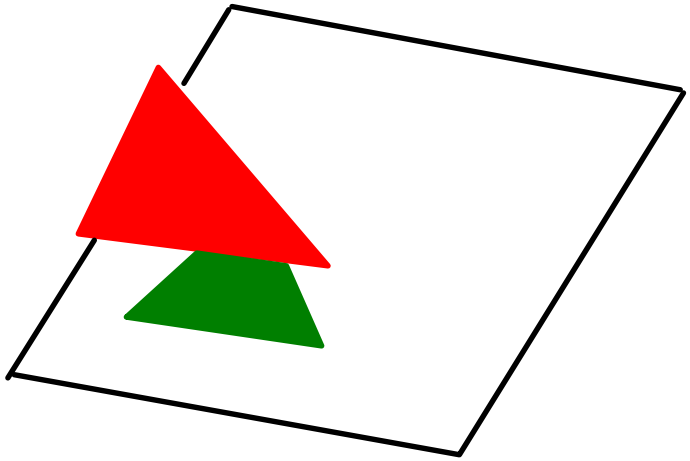


3. Does the polygon P lie completely behind the plane induced by the polygon Q (or vice versa)?

Priority algorithms

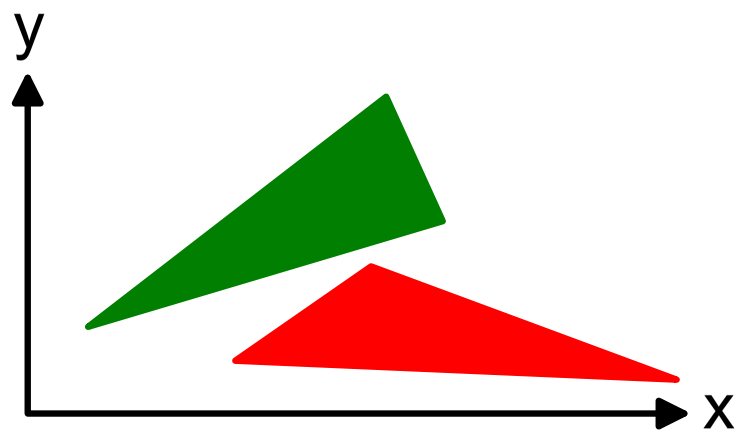


4. Does the polygon Q lie completely in front of the plane induced by the polygon P (or vice versa)?



Priority algorithms

5. Do the projections of the polygons to the (x, y)-plane not overlap?



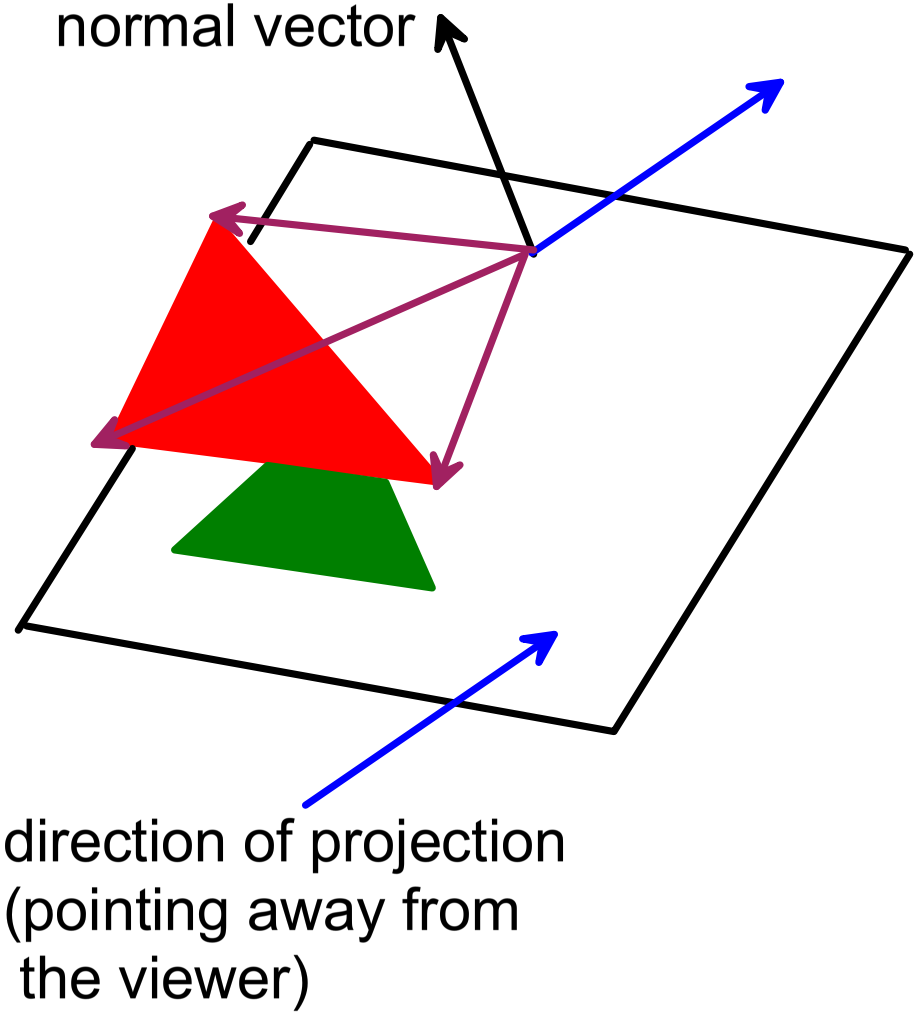
When one of these five questions has a positive answer, P is projected before Q (respectively Q before P).

Otherwise, the polygons must be further subdivided.

Priority algorithms

- 1. and 2. can be answered by comparing the x- and y-coordinates of the vertices.
- 3. and 4. can be answered by considering the angles between the normal vector to the plane and vectors connection the plane with the vertices of the polygon.

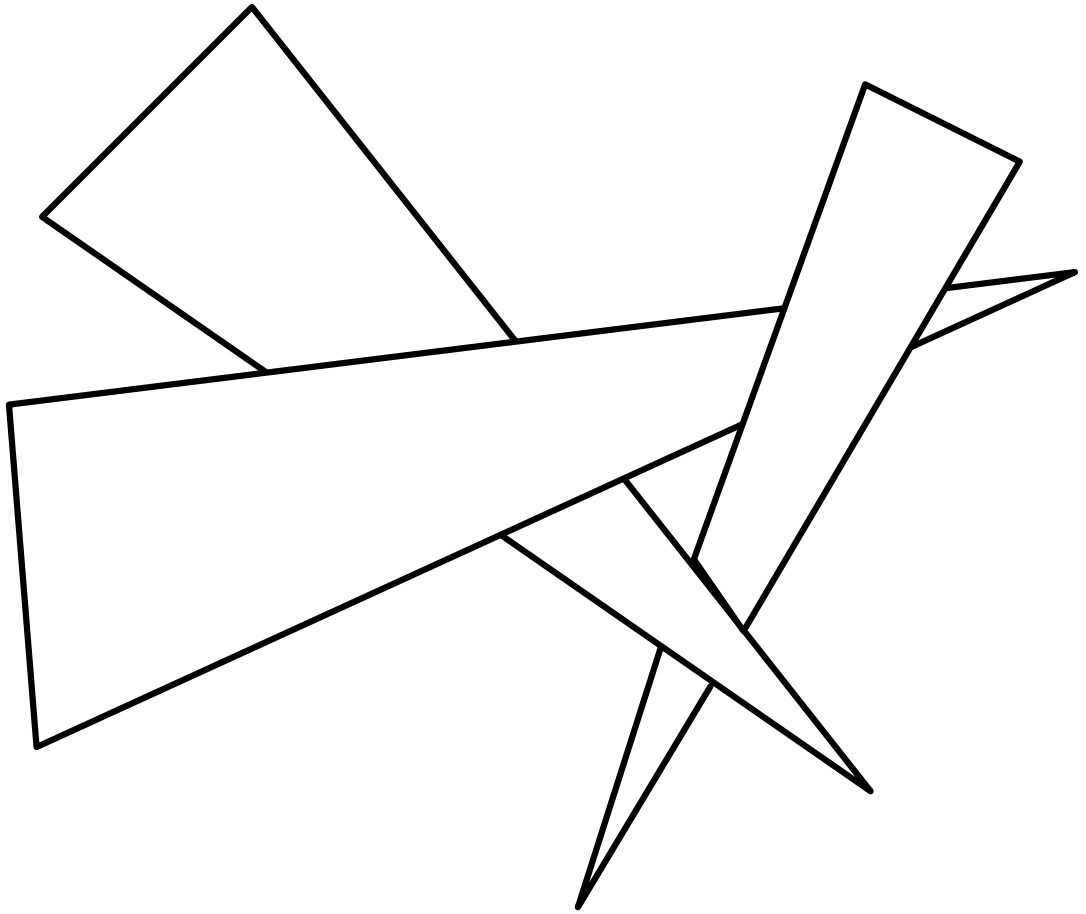
Priority algorithms



Priority algorithms

- Orientation of the normal vector such that its angle with the (blue) projection vector is larger than $90°$ (negative dot product).
- The angles between the connecting vectors to the vertices of the (red) triangle and the normal vector must all be smaller than $90°$ (positive dot product).

Priority algorithms



A case where no correct order exists in which the polygons should be projected.