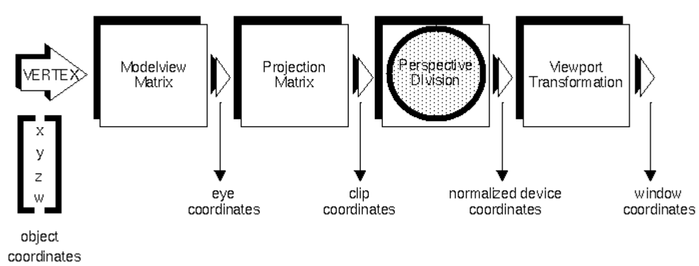
Visibility algorithms

Transformation pipeline



If several objects polygons to the same pixel how do we determine which one to draw?

Visibility problem: for each point of the screen, find the nearest surface (the one that is visible).

Visibility

Some common algorithms

- Painter's algorithms
- Z-buffer
- Ray casting

Painter's algorithm

Sort objects by depth.

Draw each object from back to front.

Advantages:

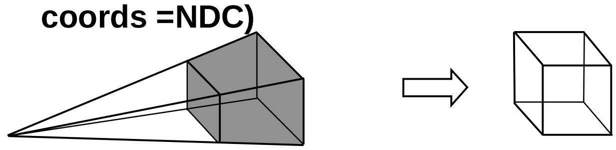
- no extra storage required
- no per-pixel operations required

Disadvantages:

- if polygons cannot be ordered they have to be split
- sorting is expensive; when the viewpoint changes, objects have to be sorted again.

Z-buffer algorithm

Recall that the transf. pipeline maps the viewing frustum into a standard cube extending from -1 to 1 in each coordinate (normalized device coords =NDC)



Instead of simply discarding the Z-coordinate in NDC during scan conversion, it is stored along with color for each pixel.

2D array of Z values = depth buffer or Z buffer

Z-buffer algorithm

After transformations the visibility order can be determined by comparing Z coordinates at each pixel. Z coordinate in NDC increases with the increase in the distance to the camera.

Algorithm:

Initialize all pixels to a large positive value.

For each polygon

For each pixel (i,j) obtained by scan converting the polygon

Find Z value for the pixel (i,j)

if Z < Zold already recorded for the pixel (i,j)

replace the color value by the color of the curent polygon and Zold with Z

Z-buffer algorithm

Advantages

- simple, suitable for hardware
- fast for scenes of moderate complexity

Disadvantages

- requires additional storage
- difficult to produce certain effects (e.g. shadows)
- time wasted drawing invisible objects

Ray Casting

fundamentally different approach:

instead of rendering each polygon or object in turn,

iterate over all pixels

For each pixel (i,j)

shoot ray from the camera position through pixel

intersect with all surfaces

find the nearest surface along the ray, use its color

Ray casting

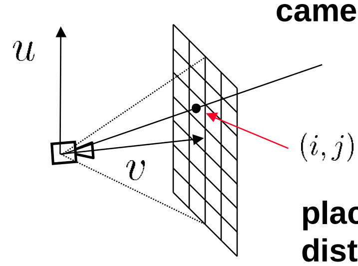
Advantages

- **Extends to ray tracing, which allows to produce many complex lighting effects**
- **Can render a variety of objects in addition to polygons**
- **Can render directly procedural objects**

Disadvantages

- **relatively slow**
- **difficult to implement in hardware**

Shooting Rays



camera model parameters:

view direction v**,**

up direction u

fov α**, aspect ratio** r

place the image plane at unit distance from the camera

pixel center position in world coords:

$$p(i,j) = c + v + \tan\frac{\alpha}{2}\left(1 - \left(i + \frac{1}{2}\right)\frac{2}{M}\right)u + r\tan\frac{\alpha}{2}\left(\frac{2}{N}\left(j + \frac{1}{2}\right) - 1\right)u \times v$$

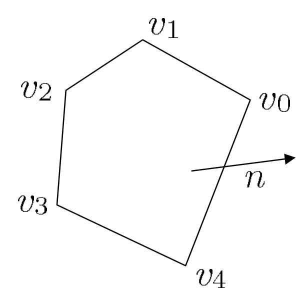
equation for the ray: $c + (p(i,j) - c)t$

M**,**N **: pixel dimensions of the image**

Intersections with polygons

Assumption: convex planar polygon with vertices

1. **Intersect with the plane.**
2. **Project the polygon and the intersection point to a coordinate plane.**
3. **Check if the projected point is inside the projected polygon.**



Intersections with polygons

Intersection with the plane of the polygon:
To find the normal, compute the cross product of any two adjacent edges that are not parallel.

<u>parametric</u> equation of the ray: $p_0 + dt,\ t \geq 0$

starting point   direction vector

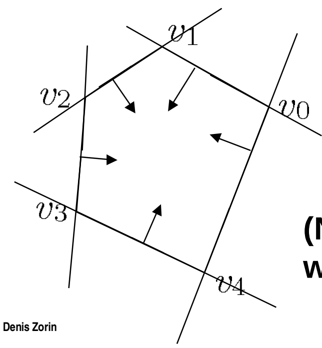
<u>implicit</u> equation of the plane: $(n, x - v_0) = 0$

intersection point position: $t_{int} = \dfrac{(n, v_0 - p_0)}{(n, d)}$

$p_{int} = p_0 + dt_{int}$ **if** $t_{int} > 0$

Intersections with polygons

Projecting the polygon and the intersection point: choose the axis for which the normal component is maximal, and project to the plane perpendicular to that axis; for example, if x component is maximal then project to yz. Projection is achieved by discarding on coordinate (x in the example).



A convex polygon can be thought of as an intersection of a set of half planes.

(Note: all vertices are 2D now! we discarded one coordinate).

Intersection with polygons

Write implicit equation for the line of each edge:

$$(d^x, d^y) = d_i = v_{i-1} - v_{i-1}$$

$$d_i^{\perp} = (-d_y, d_x) \quad (d_i^{\perp}, q - v_i) = 0$$

Check that the intersection point is one one side of all edges,

$(d_i^{\perp}, p_{int} - v_i)$ **should be positive, if vertices are enumerated counterclockwise**

Intersection with quadrics

A quadric is a surface defined by an implicit equation of total degree 2 in x,y,z; in general form:

$$Ax^2 + Dxy + Exz + Gx + By^2 + Fyz + Hy + Cz^2 + Iz + J = 0$$

in matrix form: $p^T Q p = 0$

$$Q = \begin{bmatrix} A & D/2 & E/2 & G/2 \\ D/2 & B & F/2 & H/2 \\ E/2 & F/2 & C & I/2 \\ G/2 & H/2 & I/2 & J \end{bmatrix} \qquad p = [x, y, z, 1]^T$$

Intersection with quadrics

Substitute the parametric equation for a ray into the equation for the quadric:

$$(p_0 + t\,d)^T Q (p_0 + t\,d) = (d^T Q d)t^2 + (2d^T Q p)t + p^T Q p = 0$$

The equation is, in general quadratic in t
(may be linear, should be considered separately!)

Two solutions: if both positive, take the smaller,
if one positive take the positive, if both negative, there is no intersection.
A single solution (grazing intersection): if it is positive use it.