Visibility algorithms

Visibility problem:
given a collection of objects in 3D and a camera, for each point \((x,y)\) in the image plane determine the closest object point that projects to \((x,y)\).

Different types of algorithms:
- process objects one by one, update visibility for all pixels covered by object (Z buffer, painter’s algorithm)
- process pixels one by one, for each pixel update visibility

Z buffer

Assumptions:
- each pixel has storage for a z-value, in addition to RGB
- all objects are “scanconvertible” (typically are polygons, lines or points)

Algorithm:
- Initialize zbuf to maximal value
- For each object
  - For each pixel \((i,j)\) obtained by scan conversion
    - If \(z_{\text{new}}(i,j) < z_{\text{buf}}(i,j)\)
      - \(z_{\text{buf}}(i,j) = z_{\text{new}}(i,j)\)
      - Write pixel\((i,j)\)

What are z values?

Z values are obtained by applying the projection transform, that is, mapping the viewing frustum to the standard cube.

Z value increases with the distance to the camera.

Z values for each pixel are computed for each pixel covered by a polygon using linear interpolation of z values at vertices.

Typical Z buffer size: 24 bits (same as RGB combined).

Painter’s algorithm

Algorithm:
- Apply projection transform to all polygons.
- Sort all polygons by z, splitting intersecting polygons along z.
- Scan convert polygons in back to front order.
- Polgons that are closer overwrite those that are further away.

Drawbacks: requires sorting and splits; pixels are overwritten many times.

Front to back version: add a bit to pixel to indicate it was written.

Ray casting/ray tracing

Iterate over pixels, not objects.
Effects that are difficult with Z-buffer, are easy with ray tracing: shadows, reflections, transparency, procedural textures and objects.

Assume image plane is placed in the virtual space (e.g. front plane of the viewing frustum).

Algorithm:
- For each pixel
  - Shoot a ray \(r\) from the camera to the pixel
  - Intersect with every object
  - Find closest intersection
Ray casting

Basic operation: intersect a ray with an object. Object types are more varied than for Z-buffer:
- polygon
- sphere
- cone
- cylinder
- general quadric
- height field
- ...

Pixel rays

Goal: Find direction of the ray to the center of the pixel (i,j). Let camera parameters be:
- c position
- α horizontal field of view
- v viewing direction
- u up direction
- s aspect ration

Then the image half-width in the "virtual world" units is:
\[ w = n \tan \frac{\alpha}{2} \]

The half-height is:
\[ h = s n \tan \frac{\alpha}{2} \]

Pixel rays

From coordinates in pixel units to virtual world coordinates in image plane:

Pixel center of pixel (i,j) in pixel coords:

Image center: \( c + vn \)

Pixel rays

Virtual world coordinates of pixel (i,j):

Image center + displacements.

Intersection with a sphere

Two questions are important:
- is there an intersection?
- where are the intersection points?

Most rays do not hit a sphere if it is small enough, so a fast "no" to the first question will speed up our calculation.

To answer the second question, we have to solve a quadratic equation. To answer the first, we do not have to.
**Intersection with a sphere**

**Question:** is there intersection?

The distance from the center of the sphere to the ray should be less than radius.

If \( R^2 > r^2 \), there is no intersection.

**Projection of a-c on b:**

\[
d = \frac{(a-c,b) \cdot b}{|b|^2}
\]

The square of length:

\[
((a-c \cdot d)^2) = \left( a - c - \frac{(a-c,b) \cdot b}{|b|^2}\right)^2 - R^2
\]

**Intersection with a sphere**

**Question:** what are the intersection points?

Plug in the parameteric ray equation into the sphere equation. Sphere equation can be written as \((a-q)^2 = r^2\), where \(a\) is the center and \(q\) is a point on the sphere.

\[
(a - c - bt)^2 = r^2
\]

\[
b^2t^2 - 2(a - c,b)t + (a - c)^2 - r^2 = 0
\]

\[
A t^2 + Bt + C = 0
\]

Solutions of this equation, if any, are the values of parameter \(t\) for the intersection points.