Hidden Surface Techniques

- Depth Sort / Painters Algorithm
  - Sorts objects with decreasing z-distance to the screen
  - Render scene from back to front
  - Problems, if objects overlap in their extents along z and in the x- and y-plane (piercing)
  - Problems if objects z-range is fully included in another objects z-range
  - Problems can be overcome with additional geometric tests
  - Method is very efficient for scenes with few objects (→ fast rendering)
    (used in some game engines)

- z-Buffer
  - Objects are rendered without specific order
  - Depth test is performed in the rasterization process
  - Depth compare for any pixel that is rendered
  - z-Buffer stores the closest distance of any object that been drawn at a pixel position
  - Implements correct occlusion
  - Is being implemented more and more in hardware =⇒ quite fast

z-Buffer Algorithm (depth buffer algorithm)

- Frame Buffer - keeps the color pixels of the rendered picture
- z-Buffer - keeps depth values for each picture element
- Initialization (normally):
  - Frame buffer is set to background color
  - z-Buffer elements are set to zero i.e. far clipping plane distance
  - biggest valid z-value represents front clipping plane distance
Occlusion testing is performed during rasterization for pixels

No object-object testing, no pre sorting

After transformation of fragments

Pixels of fragment are rasterized incl. depth interpolation

For every pixel, current depth is compared with z-buffer content

z-Buffer Algorithm : Pseudo-Code

```plaintext
var pz : integer

// initialization
for y = 0 to ymax do
  for x = 0 to xmax do
    begin
      WritePixel(x,y,BackgroundColor)
      WriteZ(x,y,0)
    end

// polygon rasterization
for each polygon do
  for each pixel in polygon's projection do
    begin
      pz = polygon's value at pixel coords (x,y)
      if pz > ReadZ(x,y) do
        begin // the current point is closer
          WriteZ(x,y,pz)
          WritePixel(x,y,polygons_color_at_xy)
        end
      end
    end
```

Z-Range

- Far clipping plane at $z = z_{\text{far}}$
- Near clipping plane at $z = z_{\text{near}}$
- Virtual screen
z-Buffer Algorithm continued: z-range considerations

- Depth buffer elements have limited number of bits
  - integer value representation
- Modeling coordinates are continuous floating point values
- FP z-Range must be “clamped” to discrete integer value range
  - discretization is subject to aliasing artifacts

Maintaining current z-values per pixel

**Scan-Line** Rasterization of Triangles

- \( z_a = z_1 + (z_2 - z_1) \cdot \frac{y_s - y_1}{y_2 - y_1} \)
- \( z_b = z_1 + (z_3 - z_1) \cdot \frac{y_s - y_1}{y_3 - y_1} \)
- \( z_p = z_a + (z_b - z_a) \cdot \frac{x_s - x_a}{x_b - x_a} \)

z-Buffer continued

- Aliasing artifacts: A visual example
z-Buffer

- Multiple Rendering without z-Buffer Refresh
- curved clipping surfaces

Culling Techniques (1)

- Culling : Rejection of faces which are not visible to the observer
- Purpose : Speeding up rendering of a scene

**Backface Culling**

- Sorts out polygons which are facing away from the observer
- Scene dynamic procedure depending on object-observer relation
- Reduces amount of polygons to be drawn
- Does not help to draw visible surface with correct occlusion

Culling Techniques (2)

**Occlusion Culling**

- Sorts out polygons or objects which are occluded by other objects in the scene
- Scene dynamic procedure depending on object-observer relation
- Depending partly on object material properties
- Reduces amount of polygons to be drawn

Problems:

- How to handle occlusion with semi transparent objects?
- How to handle polygons or objects which are obscured partially?

Example © 1998 Silicon Graphics, Inc.
http://www.sgi.com/software/opimaker/datasheet.html

```c
void pre_clip(VRT_Node *node)
{
    if (!show_clipping_solid)
    {
        glClearDepth(0.0);
        glDepthFunc(GL_ALWAYS);
        glDrawBuffer(GL_NONE);
    } else
    {
        glDrawBuffer(GL_BACK);
        glDepthFunc(GL_LESS);
    }
}

void post_clip(VRT_Node *node)
{
    if (!show_clipping_solid)
    {
        glDrawBuffer(GL_BACK);
        glDepthFunc(GL_GEQUAL);
    }
}

VRT_NodeSetPreRenderCallback(nclipper, pre_clip);
VRT_NodeSetPostRenderCallback(nclipper, post_clip);
```
**Special effects: Alpha blending - Transparency**

Blending the colors of overlapping objects using individual blending values, a transparency effect can be accomplished in rendering. The following compositing formula is usually applied:

\[
C_{out} = C_{src} \cdot A_{src} + (1 - A_{src}) \cdot C_{dst}
\]

- \(C_{out}\) = resulting color
- \(C_{src}\) = color of the fragment to draw
- \(C_{dst}\) = color of the fragment in the frame buffer
- \(A_{src}\) = alpha value of the fragment to draw

Important:
- Objects must be rendered in sorted order
- Objects that are farthest away from the observer are rendered first
- If objects are intersecting -> sorting must be performed for polygons of objects
- Sorting is performed along z-axes of the camera coordinate system

Example: Object 1 behind Object 2

Color of object 1:
- \(R = 200\)
- \(G = 100\)
- \(B = 20\)
- \(A = 1.0\)

Color of object 2:
- \(R = 100\)
- \(G = 40\)
- \(B = 200\)
- \(A = 0.7\)

Color of Background:
- \(R = 0\)
- \(G = 0\)
- \(B = 0\)
- \(A = 1.0\)

1. Object 1 over background:
- \(R_{out} = 200 \cdot 1.0 + (1.0 - 1.0) \cdot 0 = 200\)
- \(G_{out} = 100 \cdot 1.0 + (1.0 - 1.0) \cdot 0 = 100\)
- \(B_{out} = 20 \cdot 1.0 + (1.0 - 1.0) \cdot 0 = 20\)

2. Object 2 over background:
- \(R_{out} = 100 \cdot 0.7 + (1.0 - 0.7) \cdot 0 = 70\)
- \(G_{out} = 40 \cdot 0.7 + (1.0 - 0.7) \cdot 0 = 28\)
- \(B_{out} = 200 \cdot 0.7 + (1.0 - 0.7) \cdot 0 = 140\)

3. Object 2 over Object 1:
- \(R_{out} = 100 \cdot 0.7 + (1.0 - 0.7) \cdot 200 = 70 + 60 = 130\)
- \(G_{out} = 40 \cdot 0.7 + (1.0 - 0.7) \cdot 100 = 28 + 30 = 58\)
- \(B_{out} = 200 \cdot 0.7 + (1.0 - 0.7) \cdot 20 = 140 + 6 = 146\)

**Special effects: Fog/haze/water**

Use z-Buffer information to blend a fog color into the rendered image.

Content of the depth buffer visualized as gray-values. Huge z-values are coded dark.

A fog color (gray) is blended into the color buffer according to the z-buffers values.

\[
C_x(x,y) = (1 - f(Z(x,y))) \cdot C_x(x,y) + f(Z(x,y)) \cdot FC_x
\]

\[
C_y(x,y) = (1 - f(Z(x,y))) \cdot C_y(x,y) + f(Z(x,y)) \cdot FC_y
\]

\[
C_z(x,y) = (1 - f(Z(x,y))) \cdot C_z(x,y) + f(Z(x,y)) \cdot FC_z
\]

\[
f = k \cdot z \quad f = k \cdot e^z
\]

Using different fog colors, this effect can also be utilized for under-water scenarios.

**Special effects: Billboarding**

Objectives:
- Replace the geometric complexity by using photographic textures instead of explicit models
- Increase Rendering speed

Conditions:
- Objects must have axial or point symmetry

Precautions:
- Textured polygon must be rotated to dynamically adapt to the viewers current position

Examples:
- e.g. trees
Special effects: Billboarding continued

Axis of rotation

Observer

To correct for polygons attitude -> rotate about y-axis

Rotation angle: \( \alpha = \arccos(V \cdot N) \)

Special effects: Masked Textures & Billboards

Use alpha-masked textures to build up trees and other irregular shaped objects.

Texture map
Texture mask

Mapped onto a static polygon, viewed from the side.
Distortion effect.

Mapped onto a dynamic polygon which automatically aligns its attitude with the viewing direction.

Special effects: Textures animations

Use one big texture which contains many images from an animation. Dynamic modification of a polygons texture coordinates creates a video playback effect.

Special effects: Spatial Color Coding

Application of linear texture map to e.g. a relief map

Using the elevation value in every vertex to assign the texture co-ordinate

Benefits:

Color coding scheme can be modified by changing the texture
Colors do not need to be re-calculated per vertex
Geometry does not need to be modifies
Special effects: Antialiasing/Motion Blur

Render scene several times with different offsets. Accumulate resulting images with individual weights using "accumulation buffer".

Special effects: Mirrors

1. Reflect eye point $EP$ at mirror surface and get virtual eye point $VEP$
2. Render scene without mirror from $VEP$ into a memory frame-buffer
3. Use frame-buffer as a texture in the scene mapped onto the mirror object
4. Render entire scene from $EP$ into the visible frame buffer

Tricky issues:
- Calculation of $VEP$
- Calculation of texture coordinates for mirror polygon

Special effects: Voxelization

1. Using frontal and retro clipping plane and read pixel-operator

See whiteboard

Also:

Special effects: Spatial morphological operations

1. Applying a field of deformation vectors upon vertices of a geometry

Growing/shrinking:
- Defining center of gravity
- Vertex deformation vectors emerging from cog towards every vertex in the geometry
- Iterative vertex displacement along deformation vectors

Morphing: e.g. Transition from an apple into a banana
- Two sets of vertices/polygons for apple/banana
- Matching of vertices in the two data sets which are most likely associated with one another (automatically with spatial measures or manually)
- Calculation of deformation vectors between corresponding vertices
- Common Problem: Mesh topology does not suit the target configuration

Interactive deformation/modeling of objects (magnet/hammer)
- Defining a spatial arrangement of gravitation/deformation vectors
- Apply deformation vectors to geometry vertices depending on spatial distance
Special effects: Spatial Filtering

1. Surface smoothing

Identify vertex connectivity per vertex

For each vertex calculate vertex-deformation vector as an averaged sum of all adjacent edges

e.g.:

\[
\mathbf{v}_c = \frac{1}{n} \sum_{i=1}^{n} \mathbf{v}_i \quad \mathbf{v} = \text{deformation vector}
\]

\[
c_i \quad \text{index of current vertex}
\]

\[
n_i \quad \text{number of adjacent connected vertices}
\]

Mesh Decimation

Goal:
Find new optimized re-triangulation of a mesh in order to approximate a given surface.

Constraints:
Preserve certain surface properties (e.g. curvature, edges, bumps etc...)

Application typically:
Reduction of machine generated highly complex polygon models

Problems:
To find a suitable tradeoff between (low) polygon count and visual artifacts

Mesh Decimation

Strategies:
Elimination of “degenerated” polygons

Polygons which have an extreme edge length ratio:

- Polygons which have (relatively) extreme total dimension:

Determination of “normality” using statistical analysis of the entire mesh/model

Problem:
How to re-triangulate mesh after polygon elimination?

Mesh Decimation

Examples:
Short-Edge removal affects 2 triangles

Eliminating 2 vertices
Creating one new vertex

All triangles which share eliminated vertices need new vertex indexes
Mesh Decimation

Strategies:

Projection of slightly elevated, adjacent vertices reduces surface curvature and polygon count.

Identification and re-triangulation of connected coplanar polygons.

Find a central vertex, whose adjacent faces have little deviation in surface orientation

e.g. eliminate vertex and perform re-triangulation

E.g. project surrounding vertices onto shared plane

Mesh Optimization

Goal:
Find optimized mesh topology in order to reduce vertex transformation overhead

Approach:
Connected triangles share vertices. Shared vertices need only be processed once and can be cached.

E.g: Triangle strip generation

A triangle strip composed of n triangles requires $3 + n - 1 = n + 2$ vertex calculations